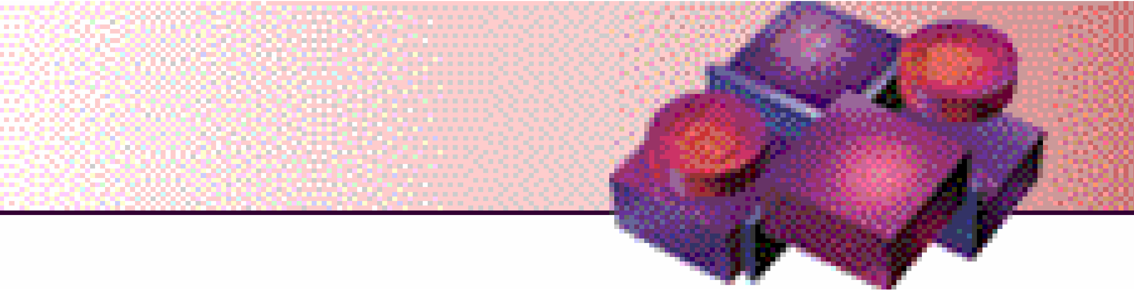


Visual Basic للجميع

نحو برمجة كائنية التوجه OOP

بقلم
تركي العسيري

بسم الله الرحمن الرحيم

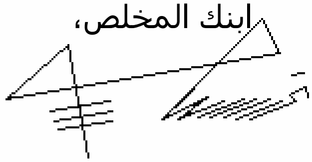
((سُبْحَانَكَ لَا عِلْمَ لَنَا إِلَّا مَا عَلَّمْتَنَا إِنَّكَ أَنْتَ الْعَلِيمُ الْحَكِيمُ))

اهداء

اهدي هذا الكتاب بباقة ورد معطرة الى الحبيبة الغالية،
التي وقف القلم حائرا عندها
محاولا ترتيب الحروف
ليكون منها كلمات
تصف شرارة من لهيب حبي لها،
والتي مهما صنعت
ما وفيت ولا كفيت في حقها
ولو بقدر اخمص قدمها...

اليك يا امي الحبيبة اهدي هذا الكتاب...

ابنك المخلص،



Visual Basic للجميع

نحو برمجة كائنية التوجه OOP

الطبعة الاولى 2002

المحتويات

الجزء الثاني: برمجة قواعد البيانات

الجزء الثالث: مواضيع متقدمة

الجزء الرابع: برمجة الانترنت

شكر وتقدير

اذا كانت الاسماء التالية لا تعني لك شيئا، فهي تعني الكثير بالنسبة لي:

في البداية اود ان اشكر جميع كتاب موقعي السابق "الى القمة مع Visual Basic" مبتدئا بعاشق الاسمبلي احمد الشمري، ومن ليبيا طارق موسى، والمبرمج المخضرم صالح الغامدي، وخبير الفلاش إياد زكري، والزميل رود ستيفن، واخي عبدالله العسيري.

كما اود ان اتقدم بالشكر الجزيل الى جميع مشرفي vb4arab.com والذي يعتبر اكبر موقع عربي يختص بالـ Visual Basic، اشكر الاساتذة: محمد الحلبي، محمد حمود، حسن الحربي، وليد عبدالله، طارق العبيد، عاصفة، صالح العنزي، محمد الدوسري، اورانوس، ساهر، جاد والمؤسس عبدالله العتيق على كل ما قدموه من جهود جبارة لمبرمجي Visual Basic العرب.

اختص بالشكر الجزيل للاستاذ سالم المالكي –مشرف عام بموقع vb4arab- على مراجعته الدقيقة واقتراحاته النيرة لهذا الكتاب. والاستاذ قاروط –مدرس مادة الـ JAVA و الـ C++ بجامعة الملك فهد للبترول والمعادن- على دروسه القوية في اساليب برمجة OOP. والاستاذ دونوفاند –استاذ في برنامج اللغة الانجليزية بالجامعة- على مقرر فن كتابة المقالات واعداد التقارير وارشادي لطريقة تأليف الكتب. وشكر جزيل الى اسطورة المبرمجين السيد بروس ماكيني –مؤلف كتاب Hardcore Visual Basic- على الاذن والسماح لي بترجمة بعض المقتطفات من كتابه وتطبيق اكواده الاحترافية.

ولا انسى شكر الزملاء فهد العمير، عبدالله القحطاني، سعد الدوسري ونايف العتيبي على اختباراتهم القوية لاكواد برامجي وتصحيح معظم اخطائها.

واذا كان شكر كل هؤلاء في كفة، فان شكر اعز واغلى البشر عندي في كفة اخرى. امي وابي اشكركما على كل ما قدمتوه لي في حياتي.

واخيرا، اتمنى ان تستمعوا بقراءة هذا الكتاب واسأل الله سبحانه وتعالى ان يجعله من العلم الذي ينتفع به انه سميع مجيب الدعوات.

المقدمة

رحلة عشر سنوات من عمر الانسان ليست كرحلة سياحية تختفي آثارها بمجرد العودة الى المكتب في العمل، فهي رحلة طويلة صاحبتها قراءات لآلاف الصفحات وسهرات بعدد النجوم التي كنت اراها بعد انصاف الليالي باسطا يدي على لوحة المفاتيح، متوغلا في صراعات مع الساحر الذي احببته وكرهته Visual Basic.

علاقة الحب والكره ليست علاقة خيالية كما تسطرها خزعبلات واساطير العشاق، فمنذ الاصدار VB1 بدأ قلبي يخفق الى تلك اللغة –مبدئيا- بسبب مصمم النماذج Form Designer الذي ساهم في نسيان اوامر المخرجات Print، Locate و Color لكتابة مئات الاسطر -اشبه بالمخطوطات الفرعونية- والخاصة بتصميم الواجهات. ولكن ما ان لبثت العلاقة الغرامية بالاشتعال، حتى بدأ شريان الكره لـ Visual Basic ينمو شيئا فشيئا بسبب قصور Visual Basic وضعفه -العاجز عن المراوغة- عند التحدث عن لغة البرمجة BASIC. الا ان علاقة الحب تطغى على علاقة الكره من جديد كلما استخدم مؤشر الفأرة Mouse عائدا لتصميم الواجهات بمصمم النماذج، وتطغى علاقة الكره مرة اخرى حين استخدام لوحة المفاتيح وكتابة الاكواد في نافذة محرر الاكواد Code Window.

وكما يقولون "الحياة تجارب"، فبعد عشر سنوات من البرمجة بـ Visual Basic، اكتشفت ان الكلمات -التي بدأت بها مقدمتي- لا اساس لها من الصحة! فـ Visual Basic يمكنك من تصميم نوافذ تضع عليها ازرار Buttons وخانات نص TextBoxes، كما يمكنك من الاتصال بقواعد البيانات DataBases لحفظ البيانات على الاقراص، وهذا بحد ذاته يلبي رغبة آلاف –ان لم يكن ملايين- المبرمجين حول العالم ليقتنعوا ان Visual Basic افضل منصة تطوير التطبيقات على مدى التاريخ.

من ناحية اخرى، لا يمكنك Visual Basic من ادارة الذاكرة Memory Management بطريقة سهلة –مقنعة للمبرمجين- وذلك بسبب ضعف تعامله مع المؤشرات Pointers، و عند الحديث عن المشاريع العملاقة، فطاقة Visual Basic لا تستوعب اكواد حجمها اكثر من K64 في نافذة نموذج واحدة او اجراء واحد، وهذا القصور يكفي ان يجعل Visual Basic اسوأ منصة تطوير التطبيقات في نظر آلاف –ان لم يكن ملايين- المبرمجين حول العالم ايضا!

إلا ان المبرمجين الموالين لـ Visual Basic قدموا عشرات الحلول لمحاولة تغطية والالتفاف حول قصور اللغة في القضايا السابقة، والمبرمجين المعادين لـ Visual Basic اثبتوا ان تلك الحلول قد أعمت القصور بدلا من تكحيلها، فهي قد اضعفت كفاءة التنفيذ وزادت الاكواد تعقيدا.

اما المبرمجين المحايدين –والمؤلف بين صفوفهم- يقولون لك بكل اختصار: اذا كان Visual Basic يقدم لك حلول لمشاكلك، فكن مبرمج Visual Basic، واذا كان Visual Basic لا يقدم لك حلول لمشاكلك، فلا تكن مبرمج Visual Basic.
ومن منطلق المحايدة، اقدم لك هذا الكتاب ليشرح لك البرمجة باستخدام Visual Basic ولن اتعدى هذا النطاق ابدا، فلا تتوقع فصول –مضيعة للوقت- تقارن لك Visual Basic مع لغات البرمجة الاخرى، او مادحة Visual Basic كلغة برمجة العقد الاخير، او مظهرة عيوب Visual Basic حتى تشوه سمعته. فهذا الكتاب يحاول تشييد بنية قوية لك حتى تمكنك من الانطلاق في برمجة Visual Basic من اوسع ابوابه.

لمن هذا الكتاب؟

اذا كنت جاد في ان تكون مبرمج Visual Basic، فهذا الكتاب يبني لك قاعدة قوية تستند عليها حتى تتمكن من مواجهة الاكواد المنتشرة هنا وهناك بين المصادر المختلفة المتعلقة بـ Visual Basic كمواقع الانترنت، مكتبة MSDN او كتب متقدمة اخرى. اما اذا كنت ناقد –غير هادف- او ترغب في الحصول على ثقافة برمجية لا تقدم ولا تؤخر، فيكفي ما قرأته من الصفحات السابقة لان الكتاب لا اعتقد انه مناسب لك.
هذا الكتاب مختص في Visual Basic فقط وليس البرمجة بشكل عام، فلن اتطرق الى مواضيع وتعاريف طويلة كـ ماهو البرنامج؟، ماهي لغة البرمجة؟، ما هو الخوارزم؟ .... الخ. مع ذلك، فهو موجه الى كآفة المستويات للاسباب التالية:

للمبتدئين: فهو يعرفهم على Visual Basic وبيئة التطوير المتكاملة الخاصة به، ويشرح لهم كل ما يحتاجونه للبرمجة الفعلية بـ Visual Basic كشرح النماذج والادوات، لغة البرمجة BASIC، حلول لمشاكلهم اليومية وزيادة ثقافتهم البرمجية بتقديم مفهوم البرمجة كائنية التوجه OOP وتطبيقها بـ Visual Basic.
للمتوسطين: فهو يحقق لهم قفزة نوعية الى مواضيع متقدمة كبرمجة قواعد البيانات DataBases، تطبيقات متقدمة على النماذج، قضايا حول اجراءات API وبرمجة المكونات COM والمكونات الموزعة DCOM.
للمتقدمين: هذا الكتاب لايقدم للمتقدمين سوى اعادة انعاش Refresh ذاكرتهم لمواضيع برمجية خاصة بـ Visual Basic، والسبب الذي يجعل هذا الكتاب موجه لهم لا يعدو ان يكون الا سبب شخصي للمؤلف، وهي زيادة الثقة بنفسه عندما يعلم ان مبرمجي Visual Basic المتقدمين قد اعجبوا في كتابه.

نقطة اخرى اود توضيحها هي ان هذا الكتاب ليس مرجع Reference من مراجع MSDN ترجم الى اللغة العربية. فلا تتوقع ان اقوم بشرح جميع الثوابت، الدوال، الكائنات .... الخ شرحا تفصيليا، وذلك لانني اريد ان انتهي من كتابة الكتاب قبل ان تشيب شعرات رأسي. مع ذلك، تعرضت –باختصار- لمئات الدوال، الخصائص، الكائنات .... الخ بين صفحات هذا الكتاب. لذلك، ارجو منك الحصول في اقرب فرصة على نسخة من اسطوانة مكتبة MSDN للحصول على التعليمات الفورية والدعم الفني اللازم للتوغل في تفاصيل جميع الدوال، الكائنات واعضائها، الثوابت .... الخ قبل ان تبدأ بقراءة الكتاب.

ماذا عن Visual Basic.NET؟

يبدو ان الحروف NET. تشد انتباه المبرمجين بعدما وزعت Microsoft النسخ التجريبية Beta من جميع اعضاء Visual Studio.NET. حسنا، الاصدار الاخير من Visual Basic هو الاصدار السادس VB6 والذي يمثل نهاية الاسطورة Visual Basic، اما Visual Basic.NET فهي لغة برمجة جديدة لا يكمن الشبه بينها وبين اسطورتنا إلا الاسم Visual Basic وصيغ Syntax بعض الاوامر. فالاسم Visual Basic.NET ليس سوى لعبة تسويقية قامت بها Microsoft حتى لا تخسر جميع زبائنها من مبرمجي VB1 حتى VB6. فقبل ان تشد الرحال الى Visual Basic.NET، فكر بالموضوع جيدا لان شد الرحال سيكون الانتقال الى لغة برمجة جديدة، كالانتقال من VB6 الى #C. ارجو ان لا تعتقد انني ارفض Visual Basic.NET، بل سأعيد كتابة الجملة الحيادية التي ذكرتها في المقدمة مع اضافة 12 حرفا واربع نقاط: اذا كان Visual Basic.NET يقدم لك حلول لمشاكلك، فكن مبرمج Visual Basic.NET، واذا كان Visual Basic.NET لا يقدم لك حلول لمشاكلك، فلا تكن مبرمج Visual Basic.NET.

اذا كنت ستستمر على احدث الاصدارات VB6، فسيكون الكتاب مناسب بتقدير جيد جدا، اما اذا كنت قد فرغت مساحة في قرصك الصلب HardDisk لتنصيب Visual Basic.NET عليها، فقد يكون الكتاب مناسب بتقدير مقبول مرتفع او جيد –على الاكثر، لان الفائدة ستكون نظرية وليس عملية، فسيتطرق هذا الكتاب الى بعض المواضيع التي قد تفيدك –نظريا- في لغتك الجديدة كالبرمجة كائنية التوجه OOP والتطبيقات متعددة الطبقات nTied Applications والمكونات الموزعة DCOM والقليل من قواعد البيانات.

مواضيع الكتاب

صفحة المحتويات اعطتك فكرة عامة عن مواضيع الكتاب، وهنا اعطيك ملخص لاجزاء الكتاب الاربعة:

الجزء الاول: الاساسيات

الغرض واضح من عنوان الجزء، فهو يبدأ بتعريفك على بيئة التطوير المتكاملة لـ Visual Basic مع كتابة برنامجك الاول، ثم يبدأ بالتحدث عن النماذج والادوات وشرح جميع خصائصها، طرقها واحداثها، ثم يقوى مهاراتك البرمجية سواء كانت في لغة البرمجة BASIC او وظائف مكتبات VB و VBA. المزيد ايضا، ينقلك هذا الجزء الى المرحلة الثانية وهي البرمجة كائنية التوجه OOP مع تفصيل مبادئها الثلاث: التغليف Encapsulation، تعدد الواجهات Polymorphism والوراثة Inheritance.

الجزء الثاني: برمجة قواعد البيانات

يعتبر هذا الجزء المدخل الرئيس الى برمجة التطبيقات المعتمدة على قواعد البيانات DataBases حيث يبدأ باساسيات طرق الوصول الى البيانات Data Access ومقرر سريع في لغة الاستعلام SQL، كما يختص كائنات ADO ويتوغل في تفاصيل بعض ادوات الجداول وانشاء التقارير.

الجزء الثالث: مواضيع متقدمة

لا تنتقل الى هذا الجزء حتى تكون قد استوعبت الجزء الاول بشكل جيد، فالجزء يتعدى افاق Visual Basic الى ان يصل الى البرمجة باستخدام اجراءات API وعرض تطبيقات عملية تؤدي الى زيادة فاعلية ومرونة برامجك، كما يحتوي على فصل بعنوان "الاستخدام المتقدم للنماذج" حيث يظهر لك امكانيات النماذج بطرق لم تخطر على بالك. واختم الجزء بفصلين لبرمجة الكائنات المركبة COM وكل ما تحتاجه لتطوير مشاريع من نوع ActiveX EXE، ActiveX DLL و ActiveX OCX مع مقدمة الى المكونات الموزعة DCOM.

الجزء الرابع: برمجة الانترنت

وهو اصغر اجزاء هذا الكتاب، فهو يحتوي على فصلين الاول يختص ببرمجة صفحات DHTML الديناميكية ومقدمة الى اللغة الصغيرة VBScript، والفصل الاخر ينقلك من برمجة العميل Client الى برمجة الخادم Server باستخدام الخادم IIS لتطوير صفحات ASP.

طريقة تنظيم الكتاب

لم استخدم سوى الطرق التقليدية لتنظيم وتنسيق محتويات الكتاب. فضلت استخدام الخط Tahoma لوضوحه ودعمه في جميع الاجهزة فهو من النوع UNICODE، وهو نفس الخط المستخدم في كتابة الاكواد، والحديث عن الاكواد في هذا الكتاب يجرني الى اخبارك انك لن تجد الاكواد المستخدمه في الامثلة الا بين صفحات هذا الكتاب، اما بعض الامثلة الطويلة او التي تتطلب ملفات متعددة لتوضيح فكرتها، فالرمز سيظهر في اعلى الكود موضحا ان الكود التالي موجود في داخل الملف المضغوط Codes.ZIP والذي تستطيع انزاله Download من نفس الموقع الذي انزلت الكتاب منه. ستلاحظ ايضا انني استخدم الاقواس [و] اذا قصدت ازرار في لوحة المفاتيح كـ [ENTER]، [ESC] ...الخ. وبالنسبة للاشكال التوضيحية، لم اكثر من استخدامها خشية ازدياد مساحة ملف الكتاب -والذي قد افصل اجزاءه الى ملفات مستقلة رغم انني لا احبذ ذلك، والاسلوب المعتمد لترقيم الاشكال يبدأ برقم الفصل ثم رقم الشكل بذلك الفصل، فلا اعتقد انك تريد البحث عن **الشكل 9485938 !**

كلمة اخيرة

بالاعتماد على مجهودي، فانه يستحيل علي كتابة ولو حرف واحد من حروف الكتاب ان لم يكتب الله سبحانه وتعالى التوفيق اولا واخيرا لاتمامه. ما اود ان اقوله لك عزيزي القارئ، ان الكتاب الذي امام عينيك قد كلفني الكثير والكثير من المجهود العقلي والبدني ومئات من ساعات عمري محاولا تقديم شئ للمكتبة العربية الالكترونية من خلاله. اذا كان التفكير في نجاح هذا الكتاب امر بعيد المدى، فحسبي ان لا يفشل، وان فشل فليس لي سوى ان أعود الى كراسي القراء باحثا عن مؤلف عربي في منصة الكتاب يرشدني الى ضالتي في مواجهة العصر الحالي لتقنية المعلومات.

اخيرا، اود ان اعتذر شديد الاعتذار عن عدم ارفاق بريدي الالكتروني على صفحات الكتاب -لاسباب شخصية- رغم انني لست ملزم بارفاقه طالما كونه كتاب مجاني للجميع، فلا تتوقع دعما فنيا مني. كما ارجو ان لا تحاول الغاء اسمي من صفحات الكتاب واستبداله باسمك او اسم شخص عزيز على قلبك، فان ذلك لا يرسم الابتسامة على شفتاي.

تركي العسيري
الظهران – يناير 2002

الجزء الاول

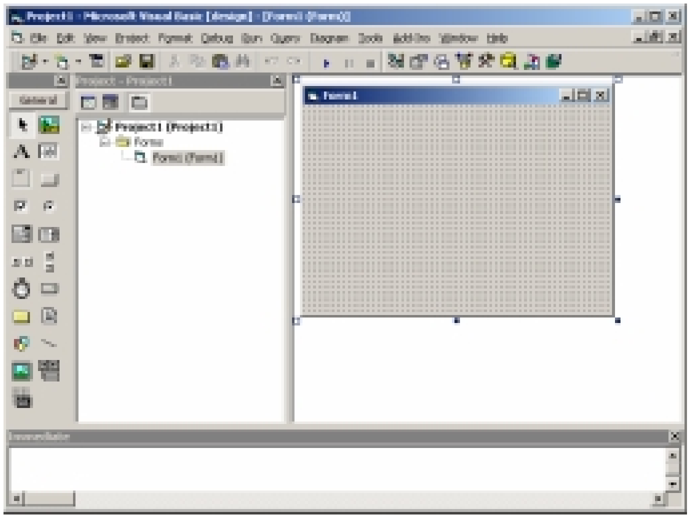
الاساسيات

الفصل الاول
تعرف على Visual Basic

بسم الله نبدأ وعلى بركته نسير مع الجملة Visual Basic. تختلف وجهات النظر بين المبرمجين حول تعريف او تصنيف Visual Basic، هل Visual Basic لغة برمجة ام برنامج تصميم نوافذ ام منتج متكامل؟ تقنيا، Visual Basic عبارة عن مكتبة من المكتبات توفر عدة اوامر برمجية متكاملة مع ابنتها Visual Basic for Application – تختصر VBA. الا ان المتعارف عليه بين المبرمجين هو ان Visual Basic لغة برمجة وبرنامج تصميم نوافذ ومكتبات تشغيلية، أي باختصار هو منتج متكامل. لذلك، عندما اتحدث عن Visual Basic في هذا الكتاب فانني اقصد جميع التعريفات السابقة، فلو ذكرت مزايا لغة Visual Basic فانني اقصد اللغة الحنونة BASIC، ولو تحدثت عن دوال واجراءات Visual Basic فانني المح الى مكتبات Visual Basic التشغيلية، ولو تطرقت الى بيئة التطوير المتكاملة لـ Visual Basic –كما في الفقرة التالية- فانني اقصد برنامج تصميم النوافذ.

بيئة التطوير المتكاملة IDE

برامجك ومشاريعك تستطيع انجازها باستخدام ابسط برنامج يأتي مع Windows والذي يدعى Notepad. الا انني –ولله الحمد- لم اقابل ذلك الشخص الذي يستخدم الـ Notepad كي يصمم برنامج ينفذ تحت Visual Basic. لان Visual Basic يوفر لك ادوات في قمة الروعة مجتمعة تحت مسمى بيئة التطوير المتكاملة Integrated Development Environment –تختصر IDE- توفر لك كل ما تحتاجه لتصميم نوافذ وكتابة اكواد برامجك. بل تقدم لك خدمات –خمس نجوم- اكثر من ذلك، كخدمة التنقيح Debugging، ادارة ملفات مشروعك، تحرير القوائم، تعديل وإنشاء قواعد البيانات ... الخ **شكل 1-1**.



شكل 1-1: بيئة التطوير المتكاملة لـ Visual Basic.

اذا كنت جديدا على هذه البيئة –أي لم تستخدم VB5- فمن الضروري التمعن في كلمات هذا الفصل بالذات والا فانك لن تدل الطرق المؤدية الى ساحة العمل. الطرق ستكون نوافذ بيئة التطوير، والعمل هو السبب الذي جعلك –على ما اعتقد- تقرأ هذا الكتاب.

كما تلاحظ في **شكل 1-1**، تحتوي بيئة التطوير على الكثير من النوافذ المحضونة بها Child Windows والعديد من القوائم Menus ومجموعات من الازرار تدعى اشرطة الادوات Toolbars، واليك يا سيدي تفاصيلها:

نوافذ بيئة التطوير المتكاملة

اذا كنت من جيل مبرمجي Visual Basic الاصدار الخامس فانتقل الى فقرة "نافذة عرض البيانات Data View" لانه لا يوجد أي شئ جديد اوضحه لك هنا. اما ان كنت من الاجيال الاقدم، فاول شئ قد يشد انتباهك هو أن النافذة الرئيسية لبيئة التطوير اصبحت من النوع متعدد المستندات Multiple Document Interface – تختصر MDI، وستلاحظ اشتمالها على نوافذ جديدة، بالاضافة إلى تطوير بعض النوافذ السابقة كنافذة مستكشف المشروع او الخصائص. واول نافذة سنبدأ بها هي من النوافذ الجديدة:

نافذة مشروع جديد New Project

عندما تقوم بتشغيل منصة العمل Visual Basic لاول مرة، فان لهذه النافذة احتمال مؤكد للظهور. فعن طريقها تستطيع الاختيار بين أنواع عدة من المشاريع كـ Standard EXE، ActiveX EXE، ActiveX DLL .... الخ. حدد النوع Standard EXE ثم انقر على الزر Open او اضغط على المفتاح [ENTER]. فالقاعدة السائدة لجميع كتب Visual Basic التعريفية تبدأ الشرح دائما بالنوع Standard EXE ولن اشذ عن هذه القاعدة، فانت لا تريد معرفة طريقة طبخ الكبسة قبل ان تتعلم قلي البيض!

اذا كانت هذه النافذة تسبب لك ازعاج -ولو بسيط- في كل مرة تشغل فيها Visual Basic، تستطيع الغاء فكرة ظهورها عن طريق تحديد الخيار Don't show this dialog in the future الموجود في اسفلها حتى لا تعكر مزاجك، وتكون رفيقا لي حتى نهاية الكتاب.

ملاحظة:	إذا لم يرق لك هذا التغيير، واردت العودة إلى الوضع السابق، فاختر الأمر ...Options من قائمة Tools، ومن مربع الحوار الذي يظهر امامك اختر تبويب Environment، وانقر على الخيار Prompt for project.

نافذة مصمم النماذج Form Designer

هذه النافذة تعتبر سر نجاح Visual Basic، وهي اشهر نوافذ Visual Basic منذ الاصدار الاول مع العنوان الابتدائي لها Form1. عن طريق هذه النافذة تقوم بعملية تصميم واجهة برنامجك اما بتعديل خصائصها او وضع باقة ادوات عليها باستخدام الفأرة Mouse مبينة لك قوة ونجاح فكرة لغات البرمجة المرئية Visual Programming Languages. فقد انتهى عصر تصميم الشاشات بالاكواد او تعليمات الماكرو Macro كما كنا نفعل في زمن اقراص 5.25 واشرطة التسجيل Tapes.

طريقة وضع الادوات على نافذة النموذج اشبه ما تكون بعملية رسم مربعات كما في برنامج الرسام Paint. المزيد ايضا، عمليات التحرير كالنسخ واللصق والقص مدعومة على جميع الادوات التي تضعها.

ملاحظة:	نافذة النموذج هي اسم مختصر من نافذة مصمم النماذج والمقصد واحد. لذلك، ستلاحظ انني استخدم المصطلح نافذة النموذج في كآفة ارجاء الكتاب .

نافذة صندوق الادوات Toolbox

بعد ان تظهر لنا نافذة النموذج السابقة، فان شـهوة وضـع الادوات عليها تصل الـى قمتها. والادوات موجـودة فـي نافـذة صـندوق الادوات Toolbox **شـكل 1-2** التـي تعرض لك 20 أداة قياسية مدمجة في جميع برامجك المصممة تحت Visual Basic، وقد تحتوي علـى مجموعـة ادوات اضـافية تـدعى ActiveX Controls -مـازال الوقـت مبكرا جدا للحديث عنها. مع ان الادوات عددها 20 أداة الا انه يوجـد 21 زر Buttons، هذا الزر الاضافي موجود في الركن العلوي الايسر مـن الادوات علـى شـكل مؤشـر Pointer وظيفته الاساسية الغاء عملية طلب رسم أداة. لا تشغل نفسك بـه كثيـرا، فهو يضغط نفسه تلقائيا بمجرد انتهائك من عملية رسـم او وضـع الأداة علـى نافـذة النموذج.



شكل 1-2: صندوق الادوات Toolbox.

اذا كـان عـدد الادوات الموجـودة فـي صـندوق الادوات كبيـرا جـدا، فيفضـل ان تقـوم بعملية تقسـيم الادوات الى مجموعـات تختفـي وتظهـر متـى مـا شـئت عـن طريـق النقر بزر الفـأرة الايمـن علـى نافـذة صـندوق الادوات واختيـار الامـر ...Add Tab مـن القائمـة المنسـدلة ومـن ثـم كتابـة اسـم المجموعـة. طريقـة تنظـيم محتويـات كـل مجموعة تتبع اسـلوب السـحب والالقـاء Drag & Drop وهـو نفـس الاسـلوب الـذي تتبعه لنسخ او نقل ملفات جهازك. اخيرا، اذا اردت حذف المجموعة، قم بالنقر علـى اسـم المجموعة بزر الفأرة الايمن واختيار الامر Delete Tab من القائمـة المنسـدلة، مع العلم ان المجموعة الرئيسـة والتي تسـمى General لن تتمكن من حذفها.
يبدو انني نسـيت نقطة مهمة وهي كيـف تعـرض نافـذة صـندوق الادوات فـي حالـة اغلاقها، يتم ذلك عن طريق اختيار الامر Toolbox من القائمة View.

نافذة الخصائص Properties Windows

بمجرد انتهائك من وضع الأداة على نافذة النموذج، فان عينيـك سـتبحث عـن موقـع نافذة الخصائص والتي من خلالها ستتمكن من تعديل خصائص الأداة او حتى نافـذة

النموذج Form، من هذه الخصائص الحجم، الموقع، اللون، العنوان .... الخ. اذا كانت هذه النافذة مخفية اختر الامر Properties Window من قائمة View او اضغط على المفتاح [F4].

في اعلى النافذة يوجد قائمة تسمى في عالم Visual Basic بأداة الـ ComboBox تمكنك من تحديد الكائن او الأداة التي تود عرض خصائصها. بامكانك تحديد الأداة مباشرة بالنقر عليها -وهي على نافذة النموذج- وستلاحظ ان محتويات نافذة الخصائص قد تغيرت. المزيد ايضا، يمكنك اختيار طريقة ترتيب جدول الخصائص اما ترتيب ابجدي Alphabetic او مصنف Categorized. وبالنسبة للجدول، فان العمود الايسر يعرض لك الخصائص المتوفرة للأداة اما الايمن فيعرض قيمة كل خاصية من هذه الخصائص. بعض الخصائص تستطيع تعديلها مباشرة بكتابة قيمة عددية او حرفية كـ Caption، وبعضها عليك اختيار قيمة من عدة قيم كـ Visible او لون من مجموعة لوح الالوان كـ BackColor، وهناك نوع يظهر لك مربع صغير في اقصى يمين العمود مكتوب عليه ثلاث نقاط "..." يقصد به صندوق حوار Dialog Box له خيارات اضافية كالخاصية Font.

نافذة مستكشف المشروع Project Explorer

تزداد اهمية هذه النافذة بازدياد عدد الملفات التابعة لمشروعك، فهي الوسيلة الوحيدة التي تمكنك من عرض محتويات مشروعك مرتبة على شكل شجري برموز مختلفة تجد شرحا مفصلا لها في ملف التعليمات. تستطيع الوصول الى الصفحة التي اقصدها عن طريق تحديد النافذة ومن ثم الضغط على المفتاح [F1]. ان لم تكن نافذة مستكشف المشروع ظاهرة امامك، تستطيع عرضها باختيار الامر Project Explorer من قائمة View او الضغط على المفاتيح [Ctrl +R].

نافذة محرر الاكواد Code Window

بما ان Visual Basic لغة برمجة، فبكل تأكيد عليك كتابة اكواد وتعليمات اللغة. عن طريق نافذة محرر الاكواد يمكنك عمل ذلك، فهي توفر لك محرر برمجي ذكي جدا ومنسق كلمات يفتح نفس المبرمج لكتابة الاكواد. من المناسب ان انوه هنا بان بعض المبرمجين العرب الذين يستخدمون نظم تدعم مجموعة محارف الشيفرة الموحدة Unicode كـ Windows 2000, XP يواجهون احيانا مشاكل في كتابة الحروف العربية، والسبب خارج نطاق الفقرة لكن حلها يتم عن طريق تغيير نوع الخط Font الي خط يدعم اللغة العربية كـ Courier New (Arabic). يمكنك عمل ذلك بالتوجه الى خانة التبويب Editor Format الموجودة في صندوق الحوار Options والذي تصل اليه عن طريق اختيار الامر Options من قائمة Tools.

بامكانك مشاهدة نافذة محرر الاكواد اما بالنقر المزدوج بالفأرة على الأداة او نافذة النموذج، او بالضغط على المفتاح [F7]. اخيرا، اذا كنت لا تحب مشاهدة افلام الرعب البرمجي المتمثلة في الاكواد الطويلة، تمكنك هذه النافذة من عرض اجراء معين واخفاء سائر الاكواد عن طريق النقر على الزر Procedure View الموجود في الركن السفلي الايسر من النافذة، لكن عملية التنقل بين الاجراءات ستكون مملة بعض الشئ عن طريق القائمتين ComboBoxes الموجودتان في اعلى النافذة، القائمة اليسرى تعرض جميع الادوات الموجودة في نافذة النموذج الحالية، بالاضافة إلى النموذج نفسة، وكذلك العبارة (General) وهي تشير إلى قسم الاعلانات في النموذج وكذلك الاجراءات والدوال التي تقوم بانشاءها، والقائمة اليمنى تعرض جميع الاجراءات والاحداث المرتبطة بما يتم اختيارة في القائمة الاولى.

نافذة مخطط النموذج Form Layout

اختيار الامر Form Layout Window الموجود في قائمة View يؤدي الى ظهور هذه النافذة والتي تعطيك رؤية مبسطة عن موقع وحجم نافذة النموذج التي تصممها وقت التنفيذ من الشاشة. الا ان الفائدة الكبرى التي تجنيها من هذه النافذة تكون مقارنة حجم نافذة النموذج مع الكثافات النقطية Resolutions المختلفة للشاشة. لعرض هذه الكثافات، انقر بزر الفأرة الايمن على النافذة واختر الامر Resolution Guide من القائمة المنسدلة.

نافذة موجة الاوامر Immediate Window

تمكنك هذه النافذة من كتابة اوامر وتعليمات لغة Visual Basic بسيطة للاختبار والتجربة، قد تحتاجها مثلا لاختبار امر معين او قيمة متغير معين قبل وضع الكود له. بعد ان تكتب الامر، المفتاح [ENTER] يؤدي الى الى تنفيذ الامر. تستطيع عرض هذه النافذة باختيار الامر Immediate Window من قائمة View.

نافذة مستعرض الكائنات Object Browser

اذا كنت مبتدءاً فإن هذه النافذة لن تثير اهتمامك، ولكن بعد ان تصل الى مرحلة متقدمة من البرمجة بـ Visual Basic سيكون لهذه النافذة تقدير كبير. تعرض لك هذه النافذة جميع الفئات الموجودة في المكتبات المضمنة في برنامجك مع كآفة طرقها، خصائصها واحداثها لتعطيك فكرة عامة عن محتويات هذه المكتبات. ميزة اخرى استخدمها كثيرا هي انها تسهل عليك عملية ايجاد صفحة التعليمات الخاصة بالامر الذي تريده عن طريق النقر بزر الفأرة الايمن على العنصر المطلوب واختيار الامر Help من القائمة المنسدلة. تستطيع عرض نافذة مستعرض الكائنات باختيار الامر Object Browser من قائمة View او الضغط على المفتاح [F2].

نافذة المتغيرات المحلية Local Window

الغرض الرئيسـي من هذه النافذة يظهر في وقت التنفيذ Run Time، لانها في وقـت التصميم Design Time لـيس لهـا أي وظيفـة ايجابيـة فـلا تعرضـها حتـى لا تـتزاحم النوافـذ امامـك. تعـرض لـك هـذه النافـذة قـيم جميـع المتغيـرات المحليـة Local Variables الموجـود فـي الاجـراء Procedure الحـالي والـذي يـتم تنفيـذه اذا قمـت بعملية الايقاف المؤقت Pause للبرنامج. في حالة وجود كائنات Objects محلية في الاجراء، فان اسـم الكائن سيظهر في الجدول ملاصق لعلامة الزائد "+" حتـى تقـوم بالنقر عليه ويعرض لـك جميـع الخصـائص وقيمهـا التابعـة لهـذا الكـائن. لعـرض هـذه النافذة اختر الامر Local Window من القائمة View.

ملاحظة:	وقت التنفيذ Run Time هو المرحلة التي يتم تنفيذ برنامجك فيها – بعـد الضغط علـى المفتـاح [F5]، امـا وقـت التصـميم Design Time فهي المرحلة التـي تصـمم برنامجـك بهـا. يوجـد وقـت خـاص يعـرف بالايقاف المؤقت Pause Time او القطع Break وهو باختصـار عمليـة الوقف المؤقت لتنفيذ برنامجك ولكن ليس انهائه.

نافذة المراقبة Watches Window

تمكنـك هـذه النافـذة مـن مراقبـة المتغيـرات او العبـارات التـي تضيفها بهـا. عمليـة المراقبة تكون متواصلة ودورية التحديث كوزارة الاعلام. بامكانك تجميد عملية تنفيـذ البرنامج –الايقاف المؤقت- في حال كـون قيمـة المتغيـر اصبحت صـحيحة True، او عند أي تغيير يطرأ على قيمة المتغير. الهدف من نافذة المراقبة ليس كاهداف وزراة الاعـلام، وانمـا تسـهيل عمليـة التنقـيح Debugging ومعرفـة قـيم المتغيـرات التـي تشكل نسبة 90% من اسباب شوائب Bugs البرامج. لعرض هذه النافذة اختر الامـر Watch Window مـن قائمـة View، ولاضـافة متغيـر او قيمـة لجعلهـا خاضـعة تحـت المراقبـة، اختـر الامـر ...Add Watch مـن قائمـة Debug –ولـيس View- او انقـر بـزر الفأرة الايمن على النافذة ومن ثم اختيار نفس الامر من القائمة المنسـدلة. الميـزة الموجودة في هذه النافذة تغطى على القصور الموجـود فـي النافـذة السـابقة، لان هذه النافذة تسمح باضافة المتغيرات العامة Global.

نافذة استدعاء الاجراءات Call Stack

تتشـابه هذه النافذة مع النافذتين السابقتين في أنها نوافذ خاصـة بـالتنقيح. تعـرض هذه النافذة طابور للاجراءات التي سيتم استدعائها، تفيدك هـذه النافـذة بقـوة فـي حالة اسـتخدامك للاسـتدعاءات التراجعيـة Recursion. اخيـرا، بامكانـك عـرض هـذه

النافذة وقت الايقاف المؤقت في حال وجود اجراءات منتظرة الاستدعاء عن طريق الامر Call Stack الموجود في قائمة View او الضغط على المفاتيح [Ctrl+L].

نافذة عرض البيانات Data View

تستطيع عرض هذه النافذة عن طريق اختيار الامر Data View Window من قائمة View او النقر على رمزها في شريط الادوات القياسي Standard Toolbar. تعتبر نافذة عرض البيانات احدث نافذة دخلت صندوق بيئة التطوير المتكاملة IDE، فلم تظهر الا في الاصدار الاخير من اصدارات Visual Basic وهو السادس. هذه النافذة البسيطة في مظهرها تعتبر اقوى نافذة من النوافذ التي ذكرتها! لانها –دون مبالغة- تعتبر برنامج كامل لمراقبة، إنشاء، تعديل، حذف وعرض جداول وحقول قواعد البيانات باختلاف أنواعها مثل: MS-SQL Server، MS-Access، ORACLE .... الخ.
لذلك، فانه من البديهي ان لا نضيع وقتنا ونفصلها الان، فالجزء الثاني من اجزاء الكتاب "برمجة قواعد البيانات" هو اهل للتفصيل -والتطريز اذا كنت تريد!

قوائم بيئة التطوير المتكاملة

اذا تحدثت عن كل امر من اوامر قوائم بيئة التطوير المتكاملة في هذه الفقرة، فسيكون الكتاب اشبه بمراجع MSDN. فيستحسن –لي ولك- ان يتم شرح وظيفة الامر في حين استخدامه، الا انني سأعرفك على القوائم بطريقة مبسطة:

القائمة File:

تحتوي هذه القائمة على اوامر اساسية خاصة للمشاريع بشكل عام، كإنشاء مشروع جديد، حفظ محتويات المشروع، طباعة محتويات المشروع وترجمة المشروع وتحويله الى ملف ثنائي Binary. الميزة التي اضيفت لـ VB6 هي امكانية فتح اكثر من مشروع في نسخة واحدة من البيئة، وهي ميزة تعرف بالمشاريع المتعددة Multiple Projects.

القائمة Edit:

تحوي هذه القائمة على اوامر التحرير القياسية كالقص، النسخ واللصق. بالاضافة الى اوامر خاصة بقواعد البيانات في حالة وجود قاعدة بيانات في نافذة عرض البيانات Data View. معظم الاوامر الواردة في اسفل هذه القائمة تستخدمها مع نافذة محرر الاكواد Code Window.

القائمة View:
ذكرت معظم محتوياتها في فقرة "نوافذ بيئة التطوير المتكاملة"، وبالاعتماد على نباهتك ومدى استعيابك يمكنك معرفة الغرض هذه القائمة.

القائمة Project:
معظم اوامرها خاصة بمحتويات المشاريع، فهي تمكنك من اضافة عنصر او عناصر من عناصر المشروع كنوافذ النماذج Forms، ملفات البرمجة Module، فئات Classes .... الخ. المزيد ايضا، يمكنك اضافة ادوات تحكم ActiveX Controls اضافية عن طريق الامر Components او تضمين مكتبات ActiveX DLL خارجية عن طريق الامر References.

القائمة Format:
الاوامر الموجودة في هذه القائمة خاصة بتنسيق الادوات التي تضعها على نافذة النموذج من ناحية موقعها على النافذة، فتوجد اوامر مرنة توفر عليك جهد محاذاة الادوات او توسيطها على النافذة، بالاضافة الى تغيير ترتيب ظهور الادوات أي وضع أداة فوق الكل او أداة خلف الكل. الامر الاخير Lock Controls تستخدمه اذا كنت راضيا عن تصميم الادوات وتود منع نفسك من تغيير احجامها او مواقعها عن طريق الخطأ، هذا القفل تستطيع فتحة بكل بساطة باختيار نفس الامر مرة اخرى.

القائمة Debug:
معظم اوامر التنقيح وضعت اسفل هذه القائمة. من هذه الاوامر اختيار طريقة تنفيذ البرنامج، كتنفيذ سطر واحد منه Step Into، اجراء كامل Step Over، امر سابق Step Out او التنفيذ حتى الوصول الى السطر الذي يوجد عليه مؤشر الكتابة Run to Cursor. وبالنسبة لنقاط القطع BreakPoints فهي علامات تظهر مبدئيا باللون الاحمر على سطر معين بحيث تتم عملية الايقاف المؤقت للبرنامج عند الوصول الى هذه العلامات.

القائمة Run:
عن طريق هذه النافذة البسيطة تستطيع تنفيذ البرنامج وتتمكنك من اختيار الاوامر الاخرى كالايقاف المؤقت Break او انهاء عملية تنفيذ البرنامج End. بالنسبة للامر Start with Full Compile هو مشابه لامر التنفيذ Start ولن تحتاجه الا في حالات نادرة ستجدها لاحقا في هذا الكتاب.

القائمة Query:

هـذه القائمـة جديـدة علـى مبرمجـي VB5 وهـي متـوفرة لنسـخة المحتـرفين Professional Edition والمؤسسـات Enterprise Edition للاصدار السـادس VB6. اوامر هذه القائمة غير ممكنة حتـى تنشـئ جملـة اسـتعلام SQL باسـتخدام الأداة Microsoft Query Builder.

القائمة Diagram:

ايضا هذه قائمة جديدة على مبرمجي VB5 ومتـوفرة لـنفس النسـخ المـذكورة فـي الفقرة السابقة. اوامر هذه القائمة غير ممكنة الا في حالة تعاملك مع قاعدة بيانـات SQL Server او ORACLE.

القائمة Tools:

تحتوي على اوامر مختلفـة التصـانيف كمحـرر القـوائم Menu Editor ومسـهل كتابـة الاجراءات Add Procedure وغيرها... اذا ادرت تخصيص بيئـة التطـوير المتكاملـة IDE فـالامر Options يمكنـك مـن الوصـول الـى صـندوق الحـوار Options الـذي يـوفر لـك عشرات الخيارات والخاصة بتغيير اعدادات بيئة التطوير IDE.

القائمة Add-Ins:

الاوامر الموجودة في هذه القائمة عبارة عن برامج مستقلة تسمى الاضـافات Add-Ins هدفها توفير خدمات اضافية لبيئة التطوير تزيد من مرونتها. تطوير هذا النوع مـن البرامج خارج نطاق الكتاب.

القائمة Window:

اذا كنت بحاجة الى شرح محتويات هذه القائمة، فيبدو انك جديد ليس على Visual Basic فحسـب وانما على جميع تطبيقات بيئة Windows، فافضل شرح اسـطره لـك هو بان توقف قراءة الكتاب –الايقاف المؤقت- في الحال، وتحاول التعـرف علـى بيئـة Windows ومن ثم تعود لمتابعة القراءة.

القائمة Help:

بالنسبة لمستخدمي الاصدار السادس من Visual Basic، فلن يتمكنوا من الوصـول الى التعليمات الفورية الا في حالة انزال نسـخة مـن مكتبـة Microsoft Developer Network والمألوفة بالاختصار MSDN.

اشرطة الادوات

جميع الازرار الموجودة في اشرطة الادوات منسوخة من القوائم السابقة، فلايوجد داعي لاعادة الشرح. اما الغرض من نسخها فهو تسريع عملية اختيار الامر. تستطيع التحكم في اشرطة الادوات وتحريرها كما تفعل مع تطبيقات MS-Office بالضغط بزر الفأرة الايمن على شريط الادوات واختيار الامر Customize من القائمة المنسدلة. احب ان انوه هنا بان صندوق الادوات ToolBox قد ادرجته ضمن فقرة نوافذ بيئة التطوير، فلا تعتقد انه من فئة اشرطة الادوات رغم الشبه الكبير بينهم.

كتابة برنامجك الاول

لاشك ان الممارسة والتطبيق احد اهم عوامل تعلم لغات البرمجة، والفصل الاول من هذا الكتاب سيبدأ معك الممارسة ليس فقط لكتابة برنامجك الاول، وانما لتوضيح المراحل والخطوات السليمة لعملية بناء البرامج سواء كانت شخصية او تجارية.

الخطوة الاولى: فكرة البرنامج

فكرة البرنامج ليست لها علاقة بـ Visual Basic ولا باي لغة اخرى ولا حتى بنظام التشغيل. فمن البديهي انك قبل ان تكتب برنامج عليك معرفة ما الذي تريده من البرنامج. قد تأتيك فكرة برنامج بينما تقلب صفحات الكتاب او متأملا النجوم في وضح النهار، وقد تكون الفكرة الزامية عليك كحل مشكلة تصادفك في جهازك او مشروع تخرج جامعي.

بعد ان تخطر الفكرة في بالك وتظهر لمبة مضيئة فوق رأسك، اسحب اللمبة وركبها في الابجورة المجاورة لجهازك، حتى تركز وتقوم بعملية التخطيط لتطبيق الفكرة كبرنامج يصمم –على الارجح- بـ Visual Basic، حدد المتطلبات والمشاكل التي ستصادفك. مثل هذه الامور تندرج تحت علم هندسة البرامج Software Engineering والتحدث عنها خارج نطاق الكتاب. لذلك، ساختم هذه الفقرة بالفكرة التي سنقوم بتنفيذها وهي عبارة عن برنامج يجري عملية الضرب بين عددين.

الخطوة الثانية: إنشاء المشروع

بعد تحديد فكرة البرنامج وتوضيح المتطلبات سنبدأ بإنشاء ملفات المشروع. قم باختيار الامر New Project من قائمة File ثم حدد النوع Standard EXE. هذا النوع

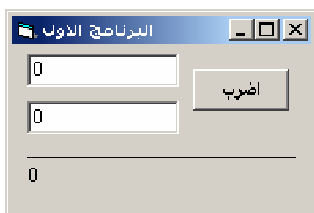
من المشاريع هي مشاريع بناء برامج قياسية تعمل تحت بيئة Windows، وعند الترجمة تكون ملفاتها من النوع EXE.
بعد قيامك بعملية إنشاء المشروع الجديد، قم بكتابة اسم مناسب للمشروع في خانة النص Project Name الموجودة في صندوق الحوار Project Properties... والذي تصل اليه باختيار الامر Project1 Properties من قائمة View. اكتب اسم ابتدائي للمشروع وليكن MyFirstProgram.

ملاحظة: يكون عنوان الامر السابق مختلف من اسم مشروع الى آخر. فبعد تعديلك لاسم المشروع، سيكون عنوان الامر MyFirstProgram Properties...

اختر الامر Save Project من قائمة File لحفظ ملفات المشروع، تذكر ان المشروع يحتوي على عدة ملفات لذلك يفضل انشاء مجلد Folder خاص بملفات المشروع قبل حفظه. ملف المشروع الرئيس يكون امتداده VBP، اما الملفات الاخرى فهي عناصر مكونة للمشروع كنوافذ النماذج FRM.*، ملفات البرمجة BAS.*، ملفات الفئات CLS.* .... الخ.

الخطوة الثالثة: تصميم الواجهة

بما ان Visual Basic يندرج تحت صنف لغات البرمجة المرئية، فعملية تصميم الواجهة تتم في الغالب باستخدام الفأرة. نبدأ في العادة بوضع الادوات على نافذة النموذج ومن ثم تعديل خصائصها. عملية وضع الادوات على نافذة النموذج تتم بشكل مشابه لرسم المربعات في برامج الرسم، واذا كنت تواجه صعوبة في عمل ذلك، فدرب نفسك بضع دقائق على برنامج الرسام Paint.
انتقل الى صندوق الادوات ToolBox وضع أداة عنوان Label و زر اوامر CommandButton واداتي TextBoxes والأداة Line لرسم الخط، ومن ثم رتب الادوات بشكل مشابه **للشكل 1-3**:



شكل 1-3: واجهة النافذة الرئيسة لبرنامجك الاول.

بعد ترتيب الادوات ومحاذاتها سنبدأ بعملية تعيين خصائصها. حدد الأداة بالنقر عليها وانتقل الى نافذة الخصائص وقم بتعديل قيم خصائص الادوات كما في الجدول التالي:

الاداة	الخاصية	القيمة
نافذة النموذج	Name	frmMain
	Caption	"البرنامج الاول"
زر الاوامر	Name	cmdMultiply
	Caption	"اضرب"
أداة العنوان	Name	lblProduct
	Caption	"0"
أداتي النص	Name	txtFirst
	Caption	"0"
	Name	txtSecond
	Caption	"0"

ملاحظة: من تقاليد مبرمجي Visual Basic كتابة حروف بادئة قبل اسم الأداة بحيث يستدل المبرمج على نوع الاداة عن طريق الأسم كـ frm لنافذة النموذج، cmd لزر الاوامر، txt لخانة النص .... الخ، وستلاحظ انني اتبعها بشكل جليل في الكتاب لتمسكي بعادات وتقاليد احزاب Visual Basic البرمجية.

وبذلك نكون قد انتهينا من الخطوة الثالثة: تصميم الواجهة.

الخطوة الرابعة: كتابة التعليمات

بعد تصميم الواجهة والاقتناع بمظهرها الفاتن، تبدأ خطوة كتابة التعليمات او الاكواد أي خطوة البرمجة الفعلية. قم بالنقر المزدوج على زر الاوامر، ستلاحظ ان نافذة محرر الاكواد Code Window قد انيرت وظهر بها هذا الكود:

```
Private Sub cmdMultiply_Click()

End Sub
```

أي كود تكتبه بين السطرين السابقين سيتم تنفيذه اذا ما قام المستخدم بالنقر على زر الاوامر، فهو مكان الحدث Click التابع للأداة cmdMultiply. وهذه فلسفة البرمجة المسيرة بالاحداث Event Driven Programming، فالاكواد لا يتم تنفيذها من اول سطر الى اخر سطر كما في السابق. كحل السطر السابق بكتابة الكود التالي الخاص باجراء عملية الضرب:

```
Private Sub cmdMultiply_Click()
	lblProduct.Caption = CDbl(txtFirst.Text) * CDbl(txtSecond.Text)
End Sub
```

ان لم تفهم الغرض من الكود السابق، فالفصول القادمة آتية اليك، وستجد فيها الجواب الشافي، ولكن اعرف الان –بشكل مؤقت- ان الكود السابق يقوم بعملية ضرب القيم التي سيكتبها المستخدم في خانات النص وسيضع ناتج الضرب في أداة العنوان.

الخطوة الخامسة: التجربة والتعديل

قم فورا بالضغط على المفتاح [F5] لتنفيذ البرنامج. ستظهر لك نافذة النموذج التي صممتها وكافة الادوات محضونة بها، اكتب الاعداد 2 و 5 في خانات النص ومن ثم انقر على الزر "اضرب" كي تحصل على الناتج 10 لامعا في أداة العنوان.
اذا كنت مسرورا جدا من نجاح عملية تنفيذ البرنامج، فتذكر ان عنوان هذه الخطوة "التجربة والتعديل" وليس "التنفيذ"، لانك ستضطر لاحقاً لاعادة تنقيح برنامجك بعدما تكتشف هذه المشكلة البسيطة اذا قام المستخدم بكتابة حروف –وليس اعداد- في خانات النص ثم قام بالنقر على الزر "اضرب"، ستلاحظ ظهور رسالة خطأ، وسيتوقف البرنامج عن العمل. يعرف هذا النوع من الأخطاء باخطاء وقت التنفيذ Run Time Errors وستختفي بهجة الفرحة باكتمال برنامجك الاول، لان Visual Basic لا يجري عملية الضرب على الحروف، فلا تتوقع ان:
تركي * 5 = تركي تركي تركي تركي تركي
لذلك، عليك باعادة عملية تعديل الكود ليتحقق من القيم المدخلة من قبل المستخدم قبل اجراء عملية الضرب عليها، ويصبح الشكل النهائي للكود المعدل هو الكود الموجود في الصفحة التالية:

```
Private Sub cmdMultiply_Click()
   ' التحقق من القيم المدخلة
   ' قبل اجراء عملية الضرب عليها
   If IsNumeric(txtFirst.Text) = True And IsNumeric(txtSecond.Text) = True Then
      lblProduct.Caption = CDbl(txtFirst.Text) * CDbl(txtSecond.Text)
   Else
      MsgBox "القيم المدخلة غير صحيحة!"
   End If
End Sub
```

الهدف من هذه الخطوة ليس شرح الاكواد او طريقة تفادي الاخطاء، وانما توضيح قضية التجربة والتعديل وبيان اهميتها قبل الانتهاء من تصميم البرنامج.

الخطوة السادسة: الترجمة

اتمنى ان تكون فرحة نجاح برنامجك قد عادت من جديد، الخطوة الاخيرة التي سنقوم بها تعرف بالترجمة Compiling، والمقصود منها عملية تحويل برنامجك الى ملف تنفيذي Executable بالامتداد EXE، يتم ذلك باختيار الامر Make MyFirstProgram.EXE من قائمة File.

ملاحظة:	في معظم لغات البرمجة الاخرى، الترجمة Compiling تكون عملية قبل عملية الربط Linking التي تنتج ملفات EXE. اما مع Visual Basic، فالترجمة Compiling هي نفس الربط Linking.

الميزة في عملية الترجمة هي امكانية تنفيذ برنامجك على جميع الاجهزة التي تحمل نظام التشغيل Windows باختلاف اصداراته والتي لا تحتوي على نسخة من Visual Basic شريطة وجود بعض مكتبات DLL التي تأتي مع Visual Basic، لعل ابرزها مكتبة MSVBVM60.DLL. لذلك، عليك ارفاق هذه المكتبة مع ملف برنامجك الرئيسي EXE الى الاجهزة الاخرى والتي لا تحتوي على Visual Basic.

الفصل الثاني
النماذج والادوات

النموذج Form هو مصطلح خاص بـ Visual Basic مرادف للنافذة Window، والنموذج هو بالفعل نافذة تقوم بتصميمها وتظهر في وقت التنفيذ كسائر نوافذ تطبيقات Windows. اما الادوات Controls فهي نوافذ ايضا، ولكن من نوع خاص توضع دائما داخل النماذج وتحتضن فيه. من المهم ان الفت النظر الى وجود نوعين من الادوات التي يمكن استخدامها مع Visual Basic، النوع الاول هي الادوات الداخلية Built-in Controls -او الجوهرية Intrinsic Controls- وهي العشرين أداة الموجودة مبدئيا في نافذة صندوق الادوات **شكل 1-2**. جميع برامجك المصممة تحت Visual Basic ملزمة بهذه الادوات حتى لو لم تستخدمها، فهي مضمنة في مكتبة MSVBVM60.DLL التي يشترط وجودها لتنفيذ برامجك. اما النوع الثاني من الادوات فتعرف بالاسم ActiveX Controls وهي ادوات خارجية يكون امتداد ملفاتها OCX. لاستخدام هذا النوع من الادوات، عليك اضافتها الى مشروعك عن طريق اختيار الامر Components من قائمة Project.
النماذج والادوات تتشارك في صفة مشتركة واحدة وهي انها كائنات Objects. وبما انها كائنات، فهي تتكون من ثلاث عناصر -تسمى الاعضاء Members- تميز الكائنات عن غيرها هي: الخصائص Properties، الطرق Methods والاحداث Events. كل كائن من كائنات Visual Basic يحتوي على اعضاء خاصة به، فالنموذج له اعضاء مستقلة وأداة النص لها اعضاء مستقلة، الا ان النماذج والادوات تحتويان على عشرات الاعضاء المشتركة بينها. لذلك، وجدت من الافضل ان ابدأ بعرض الخصائص، الطرق والاحداث المشتركة بين النماذج والادوات، ومن ثم ذكر اعضاء كل كائن على حده.

الخصائص المشتركة

الخاصية هي قيمة تؤثر اما في الشكل الخارجي للأداة كـ Caption، او في سلوك عمل الأداة كـ Enabled. نافذة الخصائص هي المكان الذي يمكنك من تغيير قيمة

الخاصية وقت التصميم، اما وقت التنفيذ فعليك كتابة الكود والذي تكون صيغته مبتدئة باسم الأداة فنقطة ثم اسم الخاصية:

```
Text1.Text = "تركي العسيري"
PictureBox1.BackColor = 0
Label1.Caption = Text1.Text
```

او استخدام الكلمة المحجوزة With اذا كنت ترغب تعديل مجموعة خصائص لكائن او اداة معينة:

```
With Text1
    .Text = "تركي العسيري"
    .Font.Bold = True
    .BackColor = vbBlack
    .ForeColor = vbWhite
End With
```

تتميز بعض الأدوات بوجود خاصية افتراضية لها تعرف بـ Default Property، تغنيك هذه الخاصية عن كتابة اسم الخاصية بعد الأداة اذا اردت تغيير قيمتها برمجياً. معظم الادوات الداخلية تكون خاصيتها الافتراضية هي Caption والخاصية Text لأداة النص:

```
Label1.Caption = "الخاصية الافتراضية"
Label1 = "الخاصية الافتراضية"
Text1 = "Text1.Text"
```

بالنسبة لنافذة النموذج، يمكنك الوصول الى خصائصها دون الالتزام بالصيغة السابقة فتستطيع استخدام الكلمة المحجوزة Me او حتى تجاهلها:

```
' جميع الاكواد التالية متشابهه
Form1.Caption = "النافذة الرئيسة"
Me.Caption = "النافذة الرئيسة"
Caption = "النافذة الرئيسة"
```

من الضروري ان انبه هنا إلى انه ليست كل الخصائص قابلة للتعديل وقت التصميم والتنفيذ، فبعض الخصائص كـ BorderStyle التابعة لنافذة النموذج لايمكنك تعديلها

وقت التنفيذ، وعلى العكس الخاصية CurrentX -التابعة لنفس الكائن- لايمكنك تعديلها الا وقت التنفيذ. المزيد ايضا، هنالك بعض الخصائص غير قابلة للتعديل مطلقا –لا في وقت التنفيذ ولا التصميم- كالخاصية hWnd، فهي خصائص تقرأها فقط وتستفيد من قيمها وهي تعرف بخصائص القراءة فقط Read Only Properties في وقت التنفيذ او التصميم او كلاهما. وعلى العكس، يوجد نوع من الخصائص تعرف بالكتابة فقط Write Only Properties ولا تستطيع قراءتها في وقت التصميم او التنفيذ او كلاهما.

والآن دعنا نتعرف اكثر على الخصائص المشتركة للادوات، والبداية ستكون مع خاصية الاسم Name.

خاصية الاسم Name

تعديل هذه الخاصية ممكن في وقت التصميم فقط، وهي الخاصية التي تمثل الاسم البرمجي للأداة. فالاسم الذي تكتبه في هذه الخاصية هو الاسم الذي تستخدمه برمجيا للوصول الى خصائصها، طرقها وحتى احداثها. وكنصيحة لا تحاول ابدا تغيير اسم الأداة بعد كتابة الكثير من الاكواد، فان ذلك سيضطرك الى تغيير جميع الاكواد للاسم السابق للأداة. نصيحة اخرى، لا تحاول الاعتماد على الاسماء الافتراضية التي يوفرها Visual Basic عند بداية رسم الأداة كـ Form1، Form2، Label1 .... الخ، فكثرة هذه الاسماء تسبب لك تشويش على ذاكرتك. كذلك لاتستطيع اختيار اسم للأداة مادام لا يحقق الشروط التالية:

- لا يبدأ برقم.
- لا يزيد عن 40 حرف.
- لا يحتوي على مسافة او علامات كـ &، ؟، " .... الخ.
- لا يكون محجوز لاسم أداة اخرى في نفس النموذج او اسم نموذج اخر في نفس المشروع –باستثناء مصفوفة الادوات كما سيأتي بيانه لاحقا.

خصائص الموقع والحجم

خصائص الموقع والحجم موجودة في جميع الادوات القابلة للظهور –أي تحتوي على الخاصية Visible، اما الادوات الاخرى كأداة المؤقت Timer فمن البديهي ان تكون هذه الخصائص غير موجودة طالما ان الأداة غير قابلة للظهور ابدا وقت التنفيذ.

خصائص الموقع Left و Top تحددان موقع الزاوية العلوية اليسرى للأداة بالنسبة الى الأداة الحاضنة لها او موقع الزاوية العلوية اليسرى لنافذة النموذج بالنسبة الى الشاشة. الوحدة المستخدمة هي نفس الوحدة المحددة في الخاصية ScaleMode التابعة للأداة الحاضنة. اما نافذة النموذج، فدائما تكون الوحدة

المستخدمة لتحديد موقعها هي الـ Twip كما سنعرف لاحقا. اذا كنت تريد توسيط الأداة وقت التصميم في نافذة النموذج، حدد الأداة ثم اختر احد الاوامر الموجودة في القائمة الفرعية Center in Form من قائمة Format، اما في وقت التنفيذ فهذا الكود يفي بالغرض:

```
Command1.Left = (Me.ScaleWidth - Command1.Width) / 2
Command1.Top = (Me.ScaleHeight - Command1.Height) / 2
```

بالنسبة لخصائص الحجم Height و Width فهي تمثل عرض وطول الأداة بنفس الوحدة المستخدمة لخصائص الموقع. في حالات معينة لن تستطيع تغيير قيمة الخاصية Height لبعض الادوات كأداة الـ ComboBox –اذا كانت خاصيتها Style تساوي 0، فقيمة الخاصية Height هنا ستكون معتمدة على نوع وحجم الخط المستخدم في الخاصية Font التابعة لها. وبعض الادوات –كأداة ListBox- فإن التحكم في خاصيتها Height ليس دقيق تماما، فقيمة هذه الخاصية تمثل عدد السطور x ارتفاع كل سطر، فلا تتوقع ظهور جزء من سطر لان سطور النصوص الموجودة فيها اما أن تعرض كاملة او لا تعرض.

خصائص الاحتضان

خصائص الاحتضان Parent و Container تمثلان مرجع Reference الى نافذة النموذج الحاضنة للأداة في الخاصية Parent او الأداة الحاضنة للأداة Container. الخاصية Parent هي للقراءة فقط –أي لا يمكنك تعديلها، اما الخاصية Container فهي قابلة للتعديل في أي وقت تريد تغيير الأداة الحاضنة للأداة:

```
' ادخال زر الاوامر داخل PictureBox
Set Command1.Container = Picture1
```

ملاحظة:	لابد من استخدام المعامل Set في الكود السابق، لانك تقوم باسناد قيم لكائنات وليس قيم عادية، الفصل الخامس سيوضح لك اسباب استخدام المعامل Set بمشيئة الله.

من المرونة التي توفرها لك هاتان الخاصيتان هي امكانية الوصول الى اعضاء –خصائص وطرق- الأداة او نافذة النموذج الحاضنة للأداة، فالكود التالي يقوم بتغيير الخاصية Caption لنافذة النموذج الحاضنة للأداة:

```
Command1.Parent.Caption = "تغير عنوان النافذة"
```

اذا كانت الأداة موجودة على نافذة النموذج ولم تحتضنها أداة اخرى، فان الخاصـيتان Parent و Container تمثلان نفس المرجع لنافذة النموذج.

خاصية الخط Font

جميع الادوات التي تعرض نصوص على جبهتها، تحتوي على هـذه الخاصـية والتـي تحدد فيها نوع وحجم الخط المعروض على جبهة الأداة. الوقت المفضل لتحديد هذه القيمة هو وقت التصميم وذلك بسبب وجود صندوق الحـوار المـألوف Font. اما فـي وقت التنفيذ، فعليك استخدام الكائن Font:

```
With Label1
        .Font.Name = "Tahoma"
        .Font.Bold = True
        .Font.Size = 20
End With
```

من المرونة التي يوفرها لك الكائن Font هي فكرة نسخ جميع خصـائص الخـط مـن أداة الى اخرى:

```
Set Label1.Font = Label2.Font
```

من المهم ان تعلم انه فـي الكـود السـابق قمنـا بنسـخ الكـائن Font التـابع لـلأداة Label2 مكان الكائن Font التابع للأداة Label1 فالكائن Font اصبح كائن واحد تـرتبط به الاداتـين، والـدليل انـه لـو قمـت بتعـديل احـد خصـائص الخـط لـلأداة Label2 فـان خاصية الخط التابعة للأداة Label1 ستتأثر ايضا:

```
' ستتأثر ايضا الأداة Label1
Label2.Font.Size = 20
```

السبب في التصرف السابق خاص بطبيعة الكائن Font الغريبة بعض الشئ.

ملاحظة: الفصل الخامس يحتوي على العديد من الامثلة العملية، ويعرض لتقنيات متقدمة في التعامل مع الكائنات بما في ذلك عملية نسخ الكائنات.

بالنسبة للخاصية Font.Name فهي تمثل اسم الخط المراد استخدامه، وفي حال استخدامك لخط غير موجود في نظام التشغيل فان Visual Basic يضع خط افتراضي من عنده، وللاسف لا استطيع ان اذكر بالتحديد ما هو الخط الذي سيستخدمة Visual Basic فهو يختلف من جهاز الى اخر. اما بالنسبة للخطوط التي لا تندرج تحت تقنية True Type، فان حجم الخط Font.Size لا يتوفر بكل المقاسات، فلا تثق بالقيمة التي ارسلتها الى هذه الخاصية، لان Visual Basic يقوم بتغييرها:

```
Label2.Font.Name = "MS SystemEx"
Label2.Font.Size = 20
Print Label2.Font.Size ` 15 الحجم
```

اخيرا، بالنسبة لمبرمجي Visual Basic المخضرمين فالخصائص القديمة FontName، FontSize، FontBold .... الخ، ما زالت مدعومة في VB6 مع ذلك، الكتاب لا يناقش خصائص غمسها التراب.

خصائص اللون

الخاصيتان BackColor و ForeColor تمثلان لون الخلفية ولون الخط للأداة. بعض الادوات كـ ScorllBar لا تدعم هاتان الخاصيتان، فألوانها تكون قياسية مستوحاه من نظام التشغيل. بعض الادوات كـ Label لا يمكن ان تلحظ التغيير في قيمة الخاصية BackColor الا اذا كانت قيمة الخاصية BackStyle تساوي 1-Opaque. كذلك الحال مع زر الاوامر CommandButton، فلن تتمكن من مشاهدة التغيير اللوني لخلفيته الا اذا حولت قيمة خاصيته Style الى 1–Graphical مع العلم ان الخاصية ForeColor ليست مدعومة فيه.

بالنسبة لقيم الالوان فمن الافضل ان اقسمها لك الى قسمين: الالوان القياسية Standard Color والالوان الخاصة Custom Color. الالوان القياسية هي المفضلة في معظم الاحوال للادوات لانها الوان يحددها المستخدم عن طريق خانة التبويب Appearance في صندوق الحوار Display Properties من لوحة التحكم، فهي الوان تناسب مزاج المستخدم ويريد ان تظهر جميع تطبيقات Windows بهذه الالوان. لذلك، من اساليب احترام ذوق المستخدم هو استخدام الالوان التي تناسبه عن

طريق الاعتماد على الالوان القياسية Standard Color. اما النوع الاخر وهو الالوان الخاصة، فهي الوان ستاتيكية أي ثابتة لا تتغير مهما قام المستخدم بتعديل خصائص سطح مكتبه، وان لم تكن مصمم راقي في اختيار الالوان المناسبة لادواتك، فانصحك بالانتقال الى فقرة "خصائص الجدولة". اما اذا كنت من المعاشرين لليوناردو دافنشي او مايكل انجلو، فتستطيع استخدام مجموعة من الثوابت تمثل ارقام الالوان:

```
Me.BackColor = vbGreen
Me.BackColor = vbBlue
```

او دالة QBColor واستخدام نفس الاعداد التي كنت تستخدمها في ايام طفولتك مع بيئة MS-DOS:

```
Me.BackColor = QBColor (0) ' اسود
Me.ForeColor = QBColor (15) ' ابيض
```

او دالة RGB مع تحديد العمق اللوني للاحمر، الاخضر والازرق:

```
Me.BackColor = RGB (255, 0, 0)
```

وإن كنت تتمتع بذاكرة قوية جدا جدا جدا تحفظ اكثر من 16 مليون لون، فتستطيع الاستفادة من هذه الذاكرة قبل الجنون واستخدم الثوابت العددية مباشرة:

```
Me.BackColor = 4234232
Me.ForeColor = &H53FF2 ' قيمة ست عشرية
```

كل الطرق السابقة تعمل بشكل جيد في وقت التنفيذ وحتى وقت التصميم، فتستطيع استخدام الدوال السابقة QBColor و RGB في خانة كتابة قيمة خصائص الالوان في جدول نافذة الخصائص، رغم انها توفر لك لوح الوان يعطيك فكرة عن عمق اللون قبل اختياره.

خصائص الجدولة

معظم مستخدمي تطبيقات Windows يفضلون استخدام مفتاح الجدولة [TAB] للتنقل بين الادوات. معظم الادوات التي لها قابلية انتقال التركيز Focus عليها –

كأداة النص، تحتوي على خصائص الجدولة TabStop و TabIndex. حدد عـن طريـق الخاصية TabStop ما اذا كنت تريد جعل التركيـز ينتقـل الـى الأداة بمجـرد ان يضـغط المسـتخدم على المفتاح [TAB]، ورتب فهرس التركيز عن طرق الخاصية TabIndex لكل أداة، مع العلم ان ترقيم الفهرس يبدأ عادة من الصفر.

ملاحظة:	حتى لو كانت قيمـة الخاصـية TabStop تسـاوي False لـلأداة، فـان المسـتخدم لديـه فرصـة اخـرى لنقـل التركيـز الـى الأداة عـن طريـق النقر عليها بزر الفأرة.

خصائص مؤشر الفأرة

خصـائص مؤشــر الفــأرة MousePointer و MouseIcon تحـددان الشــكل المطلـوب لمؤشر الفأرة Mouse Cursor. توفر لك الخاصية MousePointer 16 مؤشـر قياسـي يوفرهـا نظـام التشـغيل، وإن رغبـت فـي تخصـيص رمـز معـين، فـاختر القيمـة 99- Custom من الخاصية السـابقة مع تحميل ملـف المؤشـر فـي الخاصـية MouseIcon وقت التصميم او اكتب الكود التالي لاجراء العملية وقت التنفيذ:

```
Command1.MousePointer = vbCustom
Command1.MouseIcon = LoadPicture ("C:\Test.ICO")
```

لن تلاحظ تغيير المؤشر الا اذا مرر المسـتخدم مؤشـر الفـأرة فـوق الأداة. مـع ذلـك، هناك عدة عوامل تمنع Visual Basic من تغيير شكل المؤشـر ان تـم تغييـر المؤشـر العام للبرنامج والمتمثل في الكائن Screen، جرب هذا الكود للحظة:

```
Screen.MousePointer = 2
Command1.MousePointer = 5 ' لن يتغير شكل المؤشر ابدا
```

رغم اننا خصصنا رمز معين لزر الاوامر، الا ان Visual Basic تجاهل تخصيصنا وكأننـا لا نعنيه شيئا، التجاهل من Visual Basic ليس انقاص فـي تقـديرنا او احترامنـا، وانمـا في اسلوب تعامل Visual Basic مع خاصية MousePointer والذي يكون كالتالي:

- اذا كانت قيمة الخاصية MousePointer التابعـة للكـائن Screen غيـر 0-Default، فـان Visual Basic سـيتجاهل جميـع خصـائص MousePointer التابعـة لسـائر الادوات في البرنامج، وسيكون شكل المؤشر هو الشكل الذي تحدده في هـذه الخاصية دائما وابدا الا في حالة انتقال المؤشر الى برنامج اخر.

- اذا كانـت الخاصـية MousePointer التابعـة للكـائن Screen تسـاوي 0-Default وكانت الخاصية MousePointer التابعة للأداة لا تسـاوي 0-Default، فـان شـكل المؤشر سيكون كما هو مطلوب في الخاصية MousePointer التابعة للأداة.
- اما اذا كانت الخاصية MousePointer التابعة للكـائن Screen تسـاوي 0-Default والخاصـية MousePointer التابعـة لـلأداة تسـاوي 0-Default ايضـا، فـان شـكل المؤشر سيكون كمـا هـو مطلـوب فـي الخاصـية MousePointer التابعـة لنافـذة النموذج.

لا تقم بتغيير شكل المؤشر الا عنـد الحاجـة لتغييـره، كتحويلـه الـى صـورة يـد عنـد المرور فوق رابط لموقع على الانترنت، او على شكل الاسـهم فـي حالـة التحجـيم، ومن المسـتحسـن تحويله الى شكل ساعة رملية عند بداية كـل اجـراء حتـى يعلـم المسـتخدم ان عليه الانتظار:

```
Private Sub Command1_Click()
   ' اجراء تنفيذه يستغرق وقت
   Screen.MouseIcon = vbHourglass

   ' اكواد الاجراء
   ...

   ' لا تنسى استرجاع الشكل الافتراضي
   Screen.MousePointer = vbDefault
End Sub
```

خاصية التعريب RightToLeft

منذ VB1 حتى VB4، طال انتظار المبرمجين العرب لخاصـية –ولـو بسـيطة- تمكـنهم من تحويل اتجاه ادواتهم الى الاتجاه العربـي –مـن اليمـين الـى اليسـار، وجـاء VB5 حاملا البشرى السعيدة ليزف الـيهم الخاصـية RightToLeft المدعومـة فـي معظـم الادوات –حتـى نافـذة النمـوذج. صـحيح ان الخاصـية RightToLeft لا تطبـق تقنيـة المرآة، الا انها حلت عشـرات المشـاكل التـي كانـت تواجـه مبرمجـي Visual Basic المخضرمين.

ملاحظة: تقنية المرآة ظهرت منذ الاصدار Windows 98 وهي تقنية تقوم بقلب شكل ادوات Windows القياسية والشائعة الى الاتجاه العربي –من اليمين الى اليسار. طريقة تطبيقها تتم عن طريق الخوض في اجراءات API خاصة، سنتعرف عليها في الفصول اللاحقة بمشيئة الله.

عليك الانتباه الى ان هذه الخاصية لا تتبع مكتبة MSVBVM60.DLL وانما مكتبة خاصة بتطبيقات الشرق الاوسط تعرف بـ VBAME.DLL، الغريب في امر هذه المكتبة هو ضرورة وجودها في مجلد النظام System Directory حتى تعمل معك بشكل صحيح، فعندما تقوم بتوزيع برنامجك على اجهزة اخرى، لا تحاول وضعها في نفس مجلد البرنامج فذلك لن يفيدك. القيمة True لهذه الخاصية تحول اتجاه النافذة الى الاتجاه العربي كما تفعل ذلك مع اغلب الادوات.
اذا كان لديك نافذة نموذج مصممة وادواتها مرتبة بالاتجاه المعاكس للعربي، فهذا الكود يوفر عليك عناء اعادة ترتيب الادوات لتكون من اليمين الى اليسار:

```
Private Sub Form_Load()
   Dim Ctrl As Control
   On Error Resume Next

   For Each Ctrl In Controls
      If TypeOf Ctrl Is Line Then
         Ctrl.X1 = Ctrl.Container.ScaleWidth - Ctrl.X1
         Ctrl.X2 = Ctrl.Container.ScaleWidth - Ctrl.X2
      Else
         Ctrl.Left = Ctrl.Container.ScaleWidth - Ctrl.Left - Ctrl.Width
      End If
      If Ctrl.Alignment = 1 Then
         Ctrl.Alignment = 0
      ElseIf Ctrl.Alignment = 0 Then
         Ctrl.Alignment = 1
      End If
      Ctrl.RightToLeft = True
   Next
   RightToLeft = True
   Err.Clear
```

```
End Sub
```

خاصية المقبض hWnd

هذه الخاصية تعتبر من خصائص القراءة فقط Read Only Propeties، وهي قيمة عددية طويلة من النوع Long. حتى لو كنت من كبار المحترفين في Visual Basic، لن تستطيع الاستفادة من هذه الخاصية الا عند تعاملك مع اجراءات API والتي سنناقشها في الفصول اللاحقة. وبما ان الوقت مازال مبكرا جدا للحديث عنها، فاود توضيح نوعين من الادوات هما الادوات القياسية Standard Controls والادوات معدومة النوافذ Windowless Controls او الوهمية.

عندما تقوم بإنشاء أداة نص TextBox من صندوق الادوات، يقوم Visual Basic باجراء عملية اتصالات سرية مع نظام التشغيل Windows طالبا منه نسخة من الأداة. كرم نظام التشغيل Windows يجبره على الموافقة، ويقوم باعطاء رقم فريد لا يتكرر الى هذه الأداة يعرف بالاسم مقبض النافذة Window Handle. هذا الرقم يتم حفظه في الخاصية hWnd التابعة للأداة. من المهم ان تعلم هنا بان المسؤول الاول والاخير عن هذه الأداة هو نظام التشغيل وليس Visual Basic، فجميع العمليات التنسيقية او التي يقوم بها المستخدم يتفاعل معها نظام التشغيل وليس Visual Basic، فدور Visual Basic هنا اشبه بالمترجم بين المبرمج وبين نظام التشغيل Windows.

اما الادوات من النوع Windowless Controls فهي ادوات وهمية خاصة ببرامجك المصممة تحت Visual Basic ونظام التشغيل لا يعلم أي شئ عنها مثل الاعمى، والدليل انها لا تمتلك الخاصية hWnd.

جميع الادوات الموجودة في صندوق الادوات هي من النوع الاول باستثناء الادوات: Label، Timer، Shape، Line و Image فهي ادوات وهمية ولا تحتوي على الخاصية hWnd. حاول الاكثار من هذا النوع من الادوات فهي تستهلك القليل من مصادر النظام System Resources وتكون اسرع بكثير من الادوات الاخرى.

بالنسبة لادوات التحكم ActiveX Controls، فمعظمها من النوع Standard Controls وقد تكون من النوع Windowless Controls. ولا يمكنك معرفة نوع أداة التحكم عن طريق اختبار وجود الخاصية hWnd بها، فقد تكون الأداة من النوع الاول ولكن مصمم الأداة قد اخفى ظهور الخاصية hWnd لاسباب شخصية.

خصائص اخرى

من الخصائص المشتركة الاخرى خاصية الرؤية Visible التي تمكنك من اخفاء الأداة والادوات المحضونة بها عن عين المستخدم لكنها ظاهرة لعين المبرمج، فالأداة

موجودة في الذاكرة وبامكان المبرمج الوصول اليها حتى وان كانت مخفية. خاصية التمكين Enabled تمنع المستخدم من التفاعل مع الأداة سواء بالنقر او الكتابة عليها وهي تؤثر ايضا على الادوات المحضونة بها. الخاصية Tag تحفظ قيمة حرفية String تكون قيمة اضافية -لا راحت ولا جت- ولا تؤثر باي شكل من الاشكال على سلوك او مظهر الأداة. الخاصية Index تستخدم في حالة نسخ الادوات لتكوين ما يعرف بمصفوفة الادوات Control Array والذي سنتطرق اليه في فصل "الاستخدام المتقدم للنماذج".

ومن الخصائص التي تؤثر على مظهر الأداة والمدعومة على بعض الادوات خاصية المظهر Appearance التي تعطي مظهر ثلاثي الابعاد 3-D للأداة، والخاصية BorderStyle التي تخفي او تظهر الحدود الخارجية للأداة وأيضا خاصية المحاذاة Align التي تحاذي الأداة تلقائيا حتى مع تغيير حجم النافذة دون الحاجة الى كتابة اكواد اضافية.

تلاحظ في معظم تطبيقات Windows ظهور مربع اصفر عندما تقوم بتوجيه مؤشر الفأرة الى أداة معينة والانتظار بضع ثواني دون تحريك المؤشر، هذا المربع يدعى أداة التلميح ToolTip، بامكانك تخصيص تليمح لكل أداة موجودة في نافذة النموذج عن طريق الخاصية ToolTipText.

اخيرا، الخصائص القديمة كخصائص السحب والالقاء DragMode و DragIcon او خصائص الربط الديناميكي LinkTopic، LinkMode، LinkItem .... الخ من الخصائص التي قد جار عليها الزمن وطغت عليها تقنيات افضل منها. اذا كنت مضطر لتحقيق التوافقية Compatibility مع برامجك القديمة، فهي مازالت مدعومة بشكل جيد جدا، اما هذا الكتاب فلن ينظر الى الخلف ابدا ولن يذكر هذه الخصائص بعد النقطة التالية.

الطرق المشتركة

بعد الخصائص تأتي الطرق، الطرق عبارة عن اوامر ترسلها الى الأداة لتحريكها او نقل التركيز إليها. والواقع ان الطرق Methods هي عبارة عن دوال Functions تعود بقيم معينه، او اجراءات Sub's تقوم بوظيفة ما ولكنها لا تعيد أي قيمة. وكما توجد العديد من الخصائص المشتركة بين الادوات، توجد ايضاً عدة طرق مشتركة هي:

الطريقة Move

اذا كانت الأداة تدعم خصائص الموقع والحجم Left، Top، Height و Width، فان الطريقة Move مدعومة بها لا محالة. فالكود التالي:

```
Form1.Left = 100
Form1.Top = 200
Form1.Height = 300
Form1.Width = 400
```

يقوم يتفجير الحدث Form_Resize 4 مرات الى جانب انه يسـتهلك 4 سـطور مملـة تؤدي الى بطء في التنفيذ، من هنا تبرز ميزة الطريقة Move:

```
Form1.Move 100, 200, 300, 400
```

جميع القيم المرسلة اختيارية باستثناء القيمـة الاولـى، ولا تسـتطيع ارسـال قيمـة دون ارسال قيمة سابقة لها:

```
Form1.Move 100, 200 ' ممكن عمل ذلك
Form1.Move 100, , 300 ' انسى هذه الفكرة
```

الطريقة SetFocus

توجيه التركيز الى الأداة يـتم باسـتدعاء الطريقـة SetFocus الخاصـة بهـا. اذا كانـت الأداة مخفيـة او غيـر ممكنـة، فـان هـذه الطريقـة ستتسـبب فـي وقـوع خطـأ وقـت التشـغيل Run Time Error. لـذلك، ينصـح بـالتحقق مـن خاصـيتي الظهـور Visible والتمكين Enabled قبل نقل التركيز الى الأداة:

```
If Text1.Visible = True And Text1.Enabled = True Then
   Text1.SetFocus
End If
```

اذا كنت تريد منع المستخدم من نقل التركيز الى أي أداة اخـرى قبـل تحقـق شـرط معين، فافضل مكان هو الحدث LostFocus:

```
Private Sub Text1_LostFocus()
   If Trim(Text1.Text) = "" Then
      Text1.SetFocus
   End If
End Sub
```

اعيـد واكـرر، لا تحـاول اسـتخدام هـذه الطريقـة الا فـي حالـة ظهـور الأداة، فلـو اسـتخدمتها فـي الحـدث Form_Load مـثلا، عليـك اظهـار النافـذة قبـل اسـتخدام الطريقة:

```
Private Sub Form_Load()
    Me.Show
    Text1.SetFocus
End Sub
```

الطريقة ZOrder

قد تحتاج الى اعادة وضع أداة فـوق الادوات او خلـف الادوات وقـت التنفيـذ، الطريقـة ZOrder تفي بالغرض لوضع الأداة فـوق الادوات الاخـرى، وقـد تجعلهـا خلـف الادوات اخرى في حالة ارسال القيمة 1:

```
' القيمة الافتراضية 0
Command1.Zorder      ' فوق جميع الادوات
Command1.Zorder 0   ' فوق جميع الادوات
Command1.ZOrder 1   ' خلف جميع الادوات
```

بالنسبة للادوات معدومة النوافذ Winodwless Controls –كأداة العنوان Label- فانـه من عاشر المستحيلات ان تظهـر فـوق أداة قياسـية Standard Controls. تسـتطيع ان تفتـرض ان للنافـذة طبقتـين، الاولـى خاصـة لـلادوات القياسـية والثانيـة خاصـة لـلادوات معدومـة النوافـذ والتـي تكـون خلـف الطبقـة الاولـى دائمـا. كـذلك، الادوات الحاضـنة تكـون خلـف الادوات المحضـونة بهـا. وبالنسـبة لنوافـذ النمـاذج، فيمكنـك استخدام هذه الطريقة لوضع نافذة نموذج فـوق النوافـذ الاخـرى او خلفهـا، ولكـن لا يمكنك جعل نافذة النموذج في مقدمة نوافذ جميع تطبيقات Windows بصورة دائمة باستخدام هذه الطريقة.

الطريقة Refresh

هذه الطريقة تطلب من الأداة اعادة رسـم نفسـها. عملياً لـن تحتـاج لاسـتدعاء هـذه الطريقة كثيرا فـ Visual Basic يقوم برسـم الأداة تلقائيا بمجرد تغيير قـيم خصائصـها. الا انك قد تجبر Visual Basic لإعادة رسـم الأداة في حالات الضغط الشـديد عليه:

```
Private Sub Command1_Click()
```

```
    Dim X As Long

    For X = 0 To 100000
        Label1.Caption = CStr(X)
        Label1.Refresh
    Next
End Sub
```

قد يقترح علي احد مبرمجي Visual Basic القدماء باستخدام الدالة DoEvents. في البداية ساشكره على اقتراحه الذكي ولكن سأرفض اقتراحه هنا لان وظيفة هذه الدالة ليست مقصورة على اعادة الرسم فقط وانما تتعدى هذا الدور بكثير، فهي خاصة لعملية توزيع المعالجة Processing لباقي اجزاء البرنامج وليس الادوات فقط، مما يؤدي الى بطئ في السرعة. ليس هذا فقط، بل قد تؤدي الى شوائب برمجية Bugs، فهي تعطي فرصة كبيرة للمستخدم لاعادة الضغط على الزر Command1 مرة اخرى قبل ان ينتهي الاجراء من اكمال الحلقة التكرارية الاولى. على العموم، شكرا على الاقتراح!

الاحداث المشتركة

فلسفة البرمجة المسيرة بالاحداث Event Driven Programming تقتضي عملية تنفيذ الاكواد عند حالات معينة تعرف بوقوع الاحداث او انفجار الاحداث. فعندما تصلك رسالة امر من رئيسك في العمل، فإن استجابتك للحدث تكون بتنفيذ ما يطلب منك. كذلك الحال مع الادوات، فالاكواد التي تضعها لن يتم تنفيذها الا عند وقوع الحدث عليها. والاحداث عبارة عن اجراءات Sub's اسمائها تتبع الصيغة:

اسم الكائن_الحدث
Form_Click ()
Command1_Click ()

ملاحظة:	استخدم التعبير تفجير الحدث Fire Event عوضا عن التعبير استدعاء الحدث، فاستدعاء الحدث هي عملية كتابة اسم الحدث لتنفيذه كما تفعل مع الاجراءات، اما تفجير الحدث فهي عملية استدعاء الحدث من قبل نظام التشغيل و Visual Basic، فأرجو ان لا تتعجب من كثرة استخدامي لهذا المصطلح حتى نزول اسمي في القائمة الامريكية للمشتبه فيهم بالارهاب!

بالنسبة لنافذة النموذج، تسمية احداثها دائما ما تبدأ بالكلمة Form وليس اسم النموذج الموجود في الخاصية Name. وكما علمنا بوجود خصائص وطرق مشتركة بين الادوات، فان الاحداث لا تشذ عن هذه القاعدة:

احداث الفأرة

50% من اكوادك المستجابة تكون ردة فعل لاعمال درامية قام بها المستخدم بالفأرة. اول حدث تعرضه لك معظم الادوات عند النقر المزدوج عليها هو الحدث Click والذي ينفجر في لحظة النقر على الأداة بزر الفأرة الايسر. والحدث DblClick يمثل النقر المزدوج. لا تثق كثيرا في الحدث Click وتعتقد انه لا ينفجر الا في حالة النقر بزر الفأرة الايسر، فعند قيامك بتغيير قيمة الخاصية Value للاداتين CheckBox و OptionButton، يقوم Visual Basic تلقائيا بتفجير الحدث Click التابع للأداة. نفس الانفجار يحدث عندما تقوم بتغيير الخاصية ListIndex التابعة للاداتين ListBox و ComboBox.

من الاساليب الخاطئة التي يتبعها قليل من المبرمجين هي كتابة اكواد في كلا الحدثين Click و DblClick لنفس الأداة، رغم انك تستطيع عمل ذلك بـ Visual Basic، الا انها طريقة غير مرنة تسبب التشويش على مستخدم برنامجك تؤدي به الى الاستغناء عن الفأرة. فلو قام المستخدم بالنقر المزدوج على الأداة، فان الحدث Click سيتم تنفيذه اولا ومن ثم تنفيذ الحدث DblClick. اذا كان لابد من استخدام الحدثين في أداة واحدة، فاتمنى من صميم قلبي معرفة الحدث المقصود قبل تنفيذه حتى لا يستغني المستخدم عن فأرته:

```
Dim bDblClick As Boolean ' متغير عام

Private Sub Form_Click()
   Dim X As Single
    bDblClick = False
' اعطاء مهلة نصف ثانية
   X = Timer
   Do
      DoEvents
      If bDblClick Then Exit Sub
   Loop Until Timer > X + 0.5

' اكتب الاكواد هنا
' ...
```

```
End Sub

Private Sub Form_DblClick()
    bDblClick = True

' اكتب الاكواد هنا
' ...

End Sub
```

اذا كنت تريد معرفة المزيد من التفاصيل حول عملية النقر التي قام بها المستخدم، كموقع مؤشر الفأرة او الزر الذي استخدمه المستخدم سواء الايمن او الايسر .... الخ من تفاصيل دقيقة، فيسرني ان اعرض عليك الاحداث MouseDown، MouseMove و MouseUp التي تعطيك تفاصيل اكثر عن عمليات الفأرة على شكل متغيرات مرسلة هي: نوع الزر المستخدم Button، المفاتيح المضغوطة Shift، الاحداثي السيني للمؤشر X والاحداثي الصادي للمؤشر Y.
بالنسبة للزر المستخدم Button، فقد يكون الزر الايمن و/او الايسر و/او الاوسط للفأرة، هذا المثال يعطيك فكرة عن طريقة معرفة الازرار المضغوطة:

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, _
                 X As Single, Y As Single)
    If Button And vbLeftButton Then
        ' الزر الايسر مضغوط
    End If
    If Button And vbRightButton Then
        ' الزر الايمن مضغوط
    End If
    If Button And vbMiddleButton Then
        ' الزر الاوسط مضغوط
    End If
End Sub
```

اما المفتاح المضغوط Shift، فهو نفس المتغير Shift الخاص باحداث لوحة المفاتيح KeyUp و KeyDown والمثال التطبيقي للتعامل معه موجود في فقرة "احداث لوحة المفاتيح" التي ستصل اليها قريبا. وبالنسبة للاحداثيات X و Y فهي تمثل موقع مؤشر الفأرة بالنسبة للأداة نفسها وليس الشاشة، حيث تمثل النقطة (0، 0)

الزاوية العلوية اليسرى للأداة، وتزداد قيمة X كلما اتجه مؤشر الفأرة الى جهة اليمين وتزداد قيمة Y كلما اتجه مؤشر الفأرة الى الاسفل.
يتم تفجير الحدث MouseMove بمجرد ان يقوم المستخدم بتحريك المؤشر فوق الأداة، ونهاية الحدث تكون لحظة خروج المؤشر عن حدود الأداة. اما في حالة الالتقاط Capturing فان الحدث MouseMove سيتم تفجيره حتى لو تعدى المؤشر حدود الأداة مما يترتب عنه قيم سالبة للاحداثيات X و Y في حالة كون مؤشر الفأرة زحف يسار او فوق الأداة.

ملاحظة:	المقصد من كلمة الالتقاط Capturing هي عملية الضغط بزر الفأرة على الأداة مع استمرار الضغط على الزر.

بالنسبة للحدثين MouseDown و MouseUp فسيتم تفجيرهما بمجرد الضغط على زر الفأرة و تحرير الزر على التوالي حتى لو اختلفت الازرار، فلو قمت بالضغط على زر الفأرة الايسر –وأبقيته مضغوطا- ومن ثم قمت بالضغط على زر الفأرة الايمن، فسيقوم Visual Basic بتفجير الحدث MouseDown مرتين، وعند تحرير الازرار، فان انفجارين للحدث MouseUp مقبلان.

اخيرا، في حالة قيام المستخدم بالنقر المزدوج Double Click على الأداة، فان ترتيب وقوع او انفجار الاحداث يتم على النحو التالي:
MouseUp <- DblClick <- MouseMove <- Click <- MouseUp <- MouseDown
-> MouseMove مرة اخرى.

احداث التركيز

يتم تفجير الحدث GotFocus عندما تستقبل الأداة التركيز، والحدث LostFocus عندما تفقد الأداة التركيز، سواء كان ذلك بالفأرة او لوحة المفاتيح أو برمجيا. أما بالنسبة لنافذة النموذج، فهذه الاحداث تعمل جيدا بها شريطة عدم وجود أي أداة قابلة لاستقبال التركيز.

ملاحظة:	لن تتم عملية تفجير الاحداث بالطريقة المتوقعة اذا فقدت النافذة تركيزها بسبب الانتقال الى تطبيق اخر او استقبلت تركيزها بعد الانتقال من تطبيق اخر. باختصار، احداث التركيز لا تعمل الا بين نوافذ وادوات برنامجك فقط.

احداث لوحة المفاتيح

ثلاثة احداث مرنة يوفرها لك Visual Basic ناتجة من لوحة المفاتيح هي KeyPress، KeyDown و KeyUp. فعندما يقوم المستخدم بالضغط على أي زر من ازرار لوحة المفاتيح، فالحدث KeyDown سيتم تفجيره، ثم يقوم Visual Basic بتحويل المفتاح المدخل الى مقابله في جدول ASCII ثم يتم تفجير الحدث KeyPress، وبعد ان يرفع المستخدم اصبعه عن المفتاح يبدأ الحدث KeyUp بالانفجار.
بالنسبة للحدث KeyPress فيفجره Visual Basic في حالة قيام المستخدم الضغط على المفاتيح [ENTER]، [BACKSPACE]، [ESCAPE]، [Ctrl+...] والحروف المطبوعة، اما المفاتيح الاخرى كالاسهم او مفاتيح الوظائف وغيرها... فلا تؤدي الى انفجار الحدث KeyPress ولكن الاحداث KeyDown و KeyUp لها نصيب من الوقوع. المزيد ايضا، يرسل الحدث KeyPress قيمة عددية من النوع Integer متمثلة في متغير عددي بالاسم KeyAscii تمثل المقابل العددي للحرف المدخل في جدول ASCII:

```
Private Sub Form_KeyPress(KeyAscii As Integer)
   Print Chr$(KeyAscii) & " = " & KeyAscii
End Sub
```

المتغير KeyAscii مرسل بالمرجع وليس القيمة اي يمكنك تعديل قيمته مما يترتب عليه مرونة كبير في التحكم في مدخلات المستخدم، هذا الكود مثلا يحول جميع الحروف المدخلة في أداة النص الى حروف كبيرة Capital:

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
   KeyAscii = Asc(UCase(Chr$(KeyAscii)))
End Sub
```

واذا اسندت قيمة الصفر الى هذا المتغير، فانك قد الغيت عملية ارسال قيمة المفتاح الى الأداة المستقبلة له. هذا الكود مثلا يمنع المستخدم من كتابة أي شئ في أدا النص عدا الاعداد 0، 1، ...، 9:

```
Private Sub Text1_KeyPress(KeyAscii As Integer)
   If KeyAscii < Asc("0") Or KeyAscii > Asc("9") Then
      KeyAscii = 0
   End If
End Sub
```

ملاحظة: تلاحظ انني اعتمد في الامثلة السابقة على الدالتين Asc و Chr$. مع ذلك، يمكنك الاستغناء عنهما اذا كنت تعرف المقابل العددي للحرف المطلوب في جدول ASCII.

يزودك الحدثين KeyDown و KeyUp بقيمتين الاولى KeyCode وتمثل المفتاح المدخل، والثانية هي Shift وتمثل حالة المفاتيح [SHIFT]، [CTRL] و [ALT] فيما اذا كانت مفعوصة –اقصد مضغوطة- او لا كما في الكود التالي:

```
Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    If Shift And vbShiftMask Then
        ' المفتاح [SHIFT] مضغوط
    End If
    If Shift And vbCtrlMask Then
        ' المفتاح [CTRL] مضغوط
    End If
    If Shift And vbAltMask Then
        ' المفتاح [ALT] مضغوط
    End If
End Sub
```

بالنسبة الى قيمة المفتاح المدخل -المتمثلة في المتغير KeyCode- هي القيمة الفيزيائية للمفتاح في لوحة المفاتيح، صحيح انها مثل قيمة المتغير KeyAscii، الا انها لا تمثل نوعية الحرف المدخل سواء كان صغير Small Letter او علامات كـ ؟%#@ .... الخ، او حتى حروف عربية ا، ب، ت .... الخ. فهي ترسل دائما القيمة للحرف الانجليزي الكبير Capital كـ A، B، C .... الخ. المزيد ايضا، لا يمكننا تعديل قيمة المفتاح المدخل KeyCode كما كنا فعلنا في الصفحة السابقة مع المتغير KeyAscii.

اخيرا، احداث لوحة المفاتيح KeyPress، KeyDown و KeyUp يتم تفجيرها عندما يكون التركيز على الأداة المكتوب فيها الكود، واذا وجدت احداث اضافية تابعة لنافذة النموذج وسألتني أي الاحداث سيتم تنفيذها اولا؟، هل هي الاحداث التابعة لنافذة النموذج ام الأداة التي عليها التركيز؟ فساخبرك بان لديك عقلية نبيهة جدا جدا تستحق ان تكون مبرمج Visual Basic بسببها! فان كانت قيمة الخاصية KeyPreview التابعة لنافذة النموذج تساوي True، فان النافذة ستفجر احداثها اولا ومن ثم الأداة التي عليها التركيز، اما ان كانت قيمة هذه الخاصية False، فان نافذة

النموذج ستتجاهل هذه الاحداث وكأنها غير موجودة، ولن تفجر الا احداث الأداة فقط.

حدث التغيير Change

يتم تفجير حدث التغيير Change بمجرد القيام بتغيير محتويات الاداة كتغير النص الظاهر في الخاصية Caption او الخاصية Text. ولكن الاعتماد على هذا الحدث فيه شيء من الخطأ، فعند تغيير قيمة الخاصية Value للأداتين CheckBox و OptionButton لن يقوم Visual Basic بتفجير هذا الحدث، كذلك عند تغيير الشكل الظاهري للادوات كحجمها او الوانها لن يتم تفجير هذا الحدث.

نافذة النموذج

نافذة النموذج عزيزة على قلوب جميع مبرمجي Visual Basic، فهي البؤرة التي نرى Visual Basic من خلالها مع الاسم الابتدائي لها Form1 والذي صاحبني منذ عشر سنوات مع بدايات VB1. ولحبي لها وتقديري للعشرة الطويلة بيني وبينها، قررت تخصيص فقرة خاصة بها في هذا الفصل وفصل كامل "الاستخدام المتقدم للنماذج" في هذا الكتاب، عساها ان تميزني بين المبرمجين كما ميزتها عن سائر الكائنات!
قبل ان اخوض في تفصيل نافذة النموذج واتحدث عن خصائصها، طرقها واحداثها، بودي التطرق الى فكرة القوالب Templates او قوالب النماذج Form Templates، وهي عبارة عن نماذج جاهزة التصميم ومضبوطة الخصائص تستخدمها في برامجك اليومية بصورة متكررة دون الحاجة الى اعادة تصميمها من الصفر. اختر الامر Add Form من قائمة Project وستفهم الفكرة من قوالب النماذج الجاهزة. ففي صندوق الحوار الذي سيظهر امامك، ستجد العديد من النماذج التي تستخدمها كثيرا في برامجك الاعتيادية، واذا كانت لا تملأ بريق عينيك، صمم يا مصمم النماذج كما تريد، ومن ثم قم بحفظها في المجلد \VB98\Template\Forms -او المسارالمحدد في خانة التبويب Environment في صندوق الحوار Options- ستلاحظ وجود نافذتك كقالب Template بين القوالب السابقة.

خصائص النموذج

بعد ان تبرق نافذة النموذج امام عينيك، ستبدأ بوضع الادوات عليها ومن ثم تحجيمها. وبعد ذلك، تقوم باختيار شكل حدودها مع الخاصية BorderStyle. القيمة 0-None لا استخدمها الا في الشاشة الافتتاحية Splash Screen لبرامجي لانها

تخفي حدودها وحتى شريط عنوانها TitleBar، فتمنع المستخدم من امكانيات تحريك النافذة وتسبب له حالة من الندم على تشغيل برنامجك ان لم يقم بالضغط على المفاتيح [Ctrl+Alt+Del] او [Ctrl+Shift+Esc] كي يتمكن من اغلاق برنامجك. اما القيمة 2-Sizable فستمكن المستخدم من تحريك النافذة بانسيابية مريحة وتمكنه ايضا من اعادة تحجيم النافذة بالشكل الذي يناسبه ولن يقوم –ان شاء الله- باستخدام المفاتيح السابقة. القيمتان 1-Fixed Single و 3-Fixed Dailog تمنع المستخدم من اعادة تحجيم النافذة مع ابقاء شريط العنوان وهي قيم قياسية لصناديق حوار Dialog Boxes تطبيقات Windows، والفرق بين القيمتين يظهر جليا في ان الاولى تسمح بظهور زر التكبير Maximize والتصغير Minimize على شريط العنوان اما الثانية فلا. استخدم القيمتين 4-Fixed ToolWindow و 5-Sizable ToolWindow لتصغير ارتفاع شريط العنوان وهي موضة لنوافذ اشرطة الادوات. توجد قيمة سادسة لشكل الحد اشبه ما تكون بالقيمة 0-None لكن مع حدود ثلاثية الابعاد 3-D، ولن تستطيع مشاهدتها الا ان كانت القيمة الموجودة في الخاصية Caption خالية، وقيمة الخاصية ControlBox تساوي False، مع اختيار القيمة 3-Fixed Dailog من الخاصية السابقة BorderStyle.

تستطيع اظهار، اخفاء او تمكين صندوق التحكم او ازرار التكبير والتصغير عن طريق الخصائص ControlBox، MaxButtom و MinButton. النص الذي سيظهر في شريط عنوان النافذة هو نفس النص الموجود في الخاصية Caption. تستطيع توسيط النافذة وسط الشاشة عن طريق اختيار القيمة 2-Center من قيم الخاصية StartupPosition، وبامكانك منع المستخدم من تحريك النافذة عن طريق تحويل قيمة الخاصية Moveable الى False. اما الخاصية ShowInTaskBar فهي تضيف زر الى شريط المهام Windows Task Bar بجانب زر "ابدأ" او Start بحيث يتمكن المستخدم من تنشيط نافذة برنامجك بمجرد النقر على هذا الزر. اخيرا، خاصية WindowState التي تمكنك من تكبير النافذة لتغطي كامل الشاشة، تصغيرها او استرجاع الحجم الطبيعي لها.

خصائص الصور:

عن طريق الخاصية AutoRedraw تحدد ما اذا كانت اعادة رسم نافذة النموذج تتم تلقائيا True او يدويا False بواسطة اكوادك. في الحالة الاولى فان سرعة اعادة الرسم تكون اسرع من الحالة الثانية، الا انها تستهلك الاف الكيلوبايتات في الذاكرة مما يقلل من مصادر النظام System Resources المتاحة للبرامج الاخرى، لك ان تتخيل نافذة حجمها 800x600 مع عمق لوني True Color تستهلك 1406 كيلوبايت –ما يزيد على 1ميجا، ولك ان تتخيل 5 او 9 نوافذ جشعة من برنامجك مفتوحة، وكم ستسبب من البطء في التحميل والحجز الكبير في الذاكرة؟ من

المهم ان اذكر هنا بان الحدث Form_Paint لن يتم تفجيره ابدا طالما كانت قيمة هذه الخاصية True. باختصار، لا تستخدم هذه الخاصية الا عند الحاجة الماسة فقط، وحاول وضع اكواد الكتابة والرسم كـ Print، Line .... الخ بين سطور الحدث Form_Paint.
اذا كنت تستخدم طرق الرسم Line، Circle .... الخ بكثرة، فانصحك بتغيير قيمة الخاصية ClipControls الى False حتى تزيد سرعة طرق الرسم بمقدار الضعف لان Visual Basic لن يقوم بإنشاء منطقة Clipping region ولن يقوم باعادة الرسم الا للمناطق التي تحتاج الى اعادة رسم، اما اذا لم تستخدم طرق الرسم، فالقيمة True تكون مناسبة لهذه الخاصية.
الخاصية HasDC تخيرك فيما لو كنت تريد إنشاء سياق رسم Device Context لنافذة النموذج ام لا، سياق الرسم عبارة عن تركيب خاص بنظام التشغيل يحمل مواصفات وبيانات الصورة. اذا كنت لا تنوي وضع صورة في الخاصية Picture، فاجعل قيمة هذه الخاصية False كي تقلل من استهلاك مصادر النظام مع العلم ان الخاصية hDC لن تعمل معك الا اذا قمت بتحميل صورة على نافذة النموذج فستحمل الخاصية hDC قيمة مؤقتة تزول مباشرة بعد زوال الصورة.
الخاصية Icon تمثل الرمز الذي يظهر في صندوق التحكم Control Box التابع لنافذة النموذج والرمز الظاهر على زر النافذة في شريط المهام ، هذا إذا كانت الخاصية ShowInTaskbar تساوي True، اما ان كانت قيمة الخاصية (None) فان نظام التشغيل يضع رمز افتراضي شريطة أن تكون قيمة الخاصية ControlBox تساوي True. من الضروري أن تعلم انه لا يمكنك تخصيص رمز البرنامج EXE File Icon بشكل مستقل، فـ Visual Basic يخيرك بين احد رموز نوافذ النماذج التابعة لمشروعك عن طريق القائمة Icon من خانة التبويب Make الموجودة في صندوق حوار خصائص المشروع Project Properties.
الخاصية Picture تمكنك من تحميل ملف صورة ووضعه في داخل نافذة النموذج، تدعم هذه الخاصية هيئات مختلفة من الملفات هي: BMP، DIB، GIF، JPG، WMF، EMF، ICO و CUR. تستطيع تحميل ملف الصورة وقت التصميم باختيار اسم الملف من صندوق حوار الخاصية، او استخدام طريقة اخرى افضلها كثيرا وهي نسخ Copy الصورة من البرنامج الذي يعرضها الى الحافظة Clipboard ومن ثم لصقها باختيار الامر Paste من القائمة Edit. واذا اردت وضع الصورة في وقت التنفيذ، فالدالة LoadPicture تمكنك من فعل ذلك او سرقة صورة تابعة لكائن آخر:

```
Form1.Picture = LoadPicture ("C:\Turki.BMP") ' تحميل صورة وجهي الوسيم!
Form2.Picture = Form1.Picture ' نفس الصورة الموجودة في النموذج Form1
```

ملاحظة: الخاصية Picture هي كائن يحتوي على خصائص اضافية كعرض الصورة وارتفاع وغيرها:

```
Print Me.Picture.Height
Print Me.Picture.Width
```

اضاف VB6 متغيرات جديدة الى الدالة LoadPicture تمكنك من استخلاص رمز Icon من مجموعة رموز مضمنة في ملف ICO تجد شرح وافي لها في مكتبة MSDN، وبما انني ذكرت الدالة LoadPicture، فما المانع من ذكر زميلتها SavePicture التي تمكنك من حفظ الصورة الى ملف:

```
SavePicture Form1.Picture, "C:\Aseeri.BMP"
```

ملاحظة: الهيئة Format التي تحفظ بها الدالة SavePicture هي نفس هيئة الصورة التي حملت في الخاصية، باستثناء الهيئات GIF و JPG فيتم تحويلهما الى الهيئة BMP.

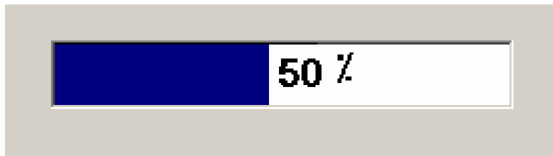
خصائص الرسم:

الخاصية Image تمثل الرسمة الموجودة على نافذة النموذج الناتجة من استخدام طرق الرسم Line، Cirle .... الخ، وستكون دائما فوق الصورة الموجودة في الخاصية Picture، لن تستطيع استخدام او الاستفادة –ان صح التعبير- من هذه الخاصية الا ان كانت قيمة الخاصية AutoRedraw هي True.

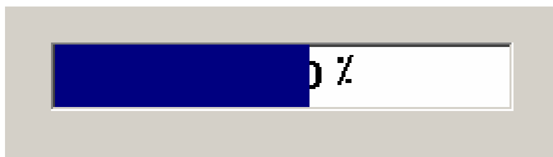
الخاصية DrawWidth تحدد عرض او سمك الفرشاة المستخدمة لرسم الخطوط والاشكال بطرق الرسم PSet، Line و Circle اما الخاصية ForeColor فهي تحدد اللون الافتراضي للطرق السابقة. بالنسبة للخاصية DrawStyle فهي تمكنك من تحديد شكل النقش لرسم الخطوط والدوائر باستخدام الطريقتين Line و Cirlce، كذلك تمكنك منه الخاصية FillStyle للمنطقة الداخلية من المربع او الدائرة مع لون التعبئة الموجود في الخاصية FillColor. اما الخاصيتان CurrentX و CurrentY فتمثلان الاحداثيات الحالية التي تستخدم لطرق المخرجات والرسم Print، Line .... الخ، واللتان تتأثران بكل عملية رسم او خرج باستخدام الطرق السابقة. اما الخاصية FontTransparent فهي تحدد اسلوب خرج الطباعة باستخدام الامر Print، فإن كانت قيمة الخاصية تساوي False فسيكون لون خلفية الطباعة هو نفس لون الخلفية BackColor للنموذج، أما إن كانت قيمة الخاصية True فإن خلفية الطباعة ستكون شفافة.

الخاصية DrawMode:

تعتبر الخاصية DrawMode من اقوى خصائص نافذة النموذج الرسومية، فعن طريقها تحدد طريقة التفاعل بين الرسوم التي ترسمها بطرق الرسم –كـ Line- مع النقاط الموجودة على نافذة النموذج. القيمة الافتراضية لهذه الخاصية هي 13-Copy Pen وتعني ان اللون سيظهر كما هو مطلوب، فالمربع الابيض سيكون ابيض ولو رسم على مربع اسود، والدائرة الحمراء سترسم حمراء ولو على سطح ارجواني. الا انك في بعض الحالات الفنية تود ان ترسم رسوم تتأثر بالالوان الموجودة على لوحة الرسم وهذا مثال واقعي تجده كثيرا في برامج التركيب **شكل 2-1**:



(ا)



(ب)



(ج)

شكل 2-1: تأثير الخاصية DrawMode على مخرجات الرسم.

تلاحظ في **الشكل 2-1 (ا)** ان المستطيل الازرق الذي رسمناه على المنطقة البيضاء قد رسم بشكل جيد جدا، ويظهر الفرق في الفن التصميمي واضحا بين الشكلين (ب) و (ج)، ففي الشكل (ب) قمنا برسم المستطيل الازرق كما نريده ازرق مما اثر وغطى على النص المكتوب "50%" ولن يتمكن المستخدم من رؤيته، اما في الشكل (ج) فقد استخدمنا القيمة المناسبة للخاصية DrawMode بحيث تقلب اللون الازرق الى ابيض في حالة الرسم فوق اللون الاسود. لمعرفة كيف تتم عملية تغيير الالوان، عليك ان تعلم ان الالوان في حقيقتها ما هي الا اعداد تتحول بالنظام الثنائي الى ارقام شبيه بـ 10011101010، اللون الذي تستخدمه يسمى

لون القلم Pen Color، واللون الموجود على لوح او سطح الرسم يسمى Screen Color، فالقيمة 15-MergePen للخاصية DrawMode تقوم بتطبيق المعادلة التالية:

S = S Or P

فلو كان اللون المستخدم هو 10101010 = 170 واللون الموجود على الشاشة هو 01010101 = 85، فان اللون الناتج –من تأثير القيمة 15-MergePen سيكون:

```
S = 01010101
P = 10101010
S = S Or P
S = 10101010 Or 01010101
S = 11111111
```

11111111 وهو 255. اذا اردت معرفة جميع المعادلات التابعة للقيم الاخرى، فمكتبة MSDN بها جدول جميل جدا تصل اليه بكتابة الجملة " DrawMode Property" في الفهرس Index.

الخاصية ScaleMode:

في بداية الفصل وبالتحديد عند فقرة "خصائص الموقع والحجم" ذكرت ان الوحدة المستخدمة لقياس احداثيات مواقع وطول وعرض الادوات هي الوحدة الموجودة في الخاصية ScaleMode. توفر لك هذه الخاصية 8 قيم تمثل وحدات Units تستخدم للقياس هي: 1-Twip والتي تعادل 0.567 سم، 2-Point تعادل 0.72 انش، 3-Pixel تعادل نقطة واحدة على الشاشة، 4-Character تعادل 120 Twips افقيا و 240 Twips عاموديا، 5-Inch تعادل انش واحد، 6-Milimeter تعادل ملم واحد، 7-Centimeter تعادل واحد سم و 0-User وحدة قياس خاصة يتم تعريفها من قبل المبرمج.

الخاصيتان Width و Height تعودان بعرض وارتفاع النافذة دائما بالوحدة Twip، فالقيمة التابعة للخاصية ScaleMode تؤثر على الوحدة المستخدمة في الادوات المحضونة فقط وليس الحاضنة، اما لمعرفة عرض وارتفاع نافذة النموذج بوحدة غير الـ Twip، قم بتحديد الوحدة في الخاصية ScaleMode واستعلم عن العرض عن طريق الخاصية ScaleWidth والارتفاع عن طريق الخاصية ScaleHeight:

```
Private Sub Form_Paint()
   Cls
   ScaleMode = vbPixels ' بالبكسل
   Print ScaleHeight
```

```
    Print ScaleWidth
End Sub
```

فـي الحقيقـة، الخاصـيتان ScaleWidth و ScaleHeight تعـودان بعـرض وارتفـاع المسـاحة الداخليـة لنافـذة النمـوذج، بينمـا تشـمل الخاصـيتان Width و Height المساحة الداخلية والخارجية المتمثلة في سمك حدودها وارتفـاع شـريط عنوانهـا. مع ذلك، لن تفرق كثيرا معـك فنـادرا مـا تحتـاج المسـاحة الخارجيـة للنافـذة، علـى العموم هذا الكود يطبع الفرق:

```
Private Sub Form_Paint()
    Cls
    ScaleMode = vbTwips
    Print Height - ScaleHeight
    Print Width - ScaleWidth
End Sub
```

اخيرا، القيمة 0-User هي وحدة تعـرف مـن قبـل المبـرمج، تسـتطيع تعريـف وحـدة خاصـة بـك عـن طريـق اسـناد قـيم الـى الخصـائص ScaleWidth، ScaleHeight، ScaleLeft و ScaleTop. قد تحتاج تعريف وحدة قياس رسـم خاصـة بـك فـي حـالات نادرة تعتمد على عرض المخططات الرسومية بشـكل استثنائي.

طرق النموذج

بما ان الفقرة السـابقة تحـدثت عـن وحـدات القيـاس التابعـة للخاصـية ScaleMode، فسأبدأ بالتحدث عن الطرق ScaleX و ScaleY. هذه الطرق تمكنك من اجراء عمليـة تحويل القياسات بين الوحدات السـابقة افقيـا وعاموديـا. ارسـل القيمـة ثـم وحـدتها الاصلية ثم الوحدة المطلوبة:

```
' التحويل من Pixels الى Twips
Print ScaleX(100, vbPixels, vbTwips)
```

الطريقة Show تؤدي الى اظهار النموذج والطريقـة Hide تخفيـه، نسـتطيع ان نقـول بكل ثقة انهما يمثلان الخاصية Visible ولكن على شكل طرق:

```
Form1.Show ' Form1.Visible = True
Form1.Hide ' Form1.Visible = False
```

طرق الرسم

الطريقة Cls تمسح جميع الرسوم الموجودة على النافذة وتصفر الاحداثيات CurrentX و CurrentY الى الاحداثي (0, 0)، والطريقة Point تعود بالقيمة العددية للون الموجود في الاحداثي (x, y) على النافذة:

```
Private Sub Form_Load()
    ' تحميل صورة وجهي الوسيم!
    Form1.Picture = LoadPicture ("C:\Turki.BMP")
End Sub

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, _
        X As Single, Y As Single)
    Label1.BackColor = Point(X, Y)
    Label1.Caption = Hex$(Point(X, Y))
End Sub
```

الطريقة PSet تمكنك من رسم نقطة على النافذة بارسال الاحداثي (X, Y) للنقطة، سيكون لون النقطة هو نفس اللون الموجود في الخاصية ForeColor او بامكانك ارسال اللون:

```
Me.ForeColor = vbBlack
PSet (0, 0)                    ` نقطة سوداء
PSet (500, 500), vbRed         ` نقطة حمراء
```

الطريقة PSet –وطرق الرسم الاخرى- تدعم الكلمة المحجوزة Step والتي تضيف الاحداثيات المرسلة (X, Y) الى الاحداثيات الحالية –الموجودة في الخاصيتان CurrentX و CurrentY:

```
Private Sub Form_Paint()
    Dim X As Integer

    Cls
    Me.CurrentX = 0
    Me.CurrentY = 0

    For X = 0 To 100
```

```
      PSet Step(5, 5)
   Next
End Sub
```

الطريقة Line تمكنك من رسم الخطوط بارسال احداثيات البداية (X1, Y1) والنهاية (X2, Y2):

```
ForeColor = vbGreen
Me.Line (0, 0) - (Me.ScaleWidth, Me.ScaleHeight)            ' خط اخضر
Me.Line (0, Me.ScaleHeight) - (Me.ScaleWidth, 0), vbRed   ' خط احمر
```

في حالة تجاهلك للاحداثي (X1, Y1) فان القيم الحالية للخصائص CurrentX و CurrentY هي نقطة البداية:

```
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, _
                X As Single, Y As Single)
   Me.Line - (X, Y)
End Sub
```

الطريقة Line تمكنك من رسم المستطيلات عوضا عن الخطوط بارسال الحرف B مع العلم ان النقش سيكون النقش المحدد في الخاصية FillStyle واللون FillColor:

```
ForeColor = vbWhite
Line (0, 0) - (200, 200), , B            ' مربع ابيض
Line – Step (200, 200), vbBlue, B    ' مربع ازرق
```

ولتلوين المربع مباشرة –دون استخدام الخصائص، ارسل الحرفين BF:

```
ForeColor = vbWhite
Line (0, 0) - (200, 200), , BF           ' مربع ابيض
Line – Step (200, 200), vbBlue, BF   ' مربع ازرق
```

واختم فقرة طرق الرسم بالطريقة Cirlce التي من الواضح انها لا ترسم نجوم! وانما دوائر بارسال احداثي نقطة مركز الدائرة ومن ثم طول قطرها:

```
ForeColor = vbWhite
Circle (Me.ScaleWidth / 2, Me.ScaleHeight / 2), 500              ' دائرة بيضاء
Circle (Me.ScaleWidth / 2, Me.ScaleHeight / 2), 200, vbGreen     ' دائرة خضراء
```

عملية تلوين الدائرة لا تتم باستخدام BF كما في الطريقة BF، وانما بالقيمة الموجودة في الخاصية FillColor مع النقش FillStyle.
من الاشياء التي تعجبني في الطريقة Cirlce هي امكانية رسم الاقواس بتحديد زاوية النهاية والبداية بوحدة الراديان Radian:

```
Const PI = 3.14
' نصف دائرة فتحهتها الى الاعلى
Circle (Me.ScaleWidth / 2, Me.ScaleHeight / 2), 1000, , 0, PI
' نصف دائرة فتحهتها الى الاسفل
Circle (Me.ScaleWidth / 2, Me.ScaleHeight / 2), 800, , PI, 0
' ربع دائرة
Circle (Me.ScaleWidth / 2, Me.ScaleHeight / 2), 500, , 0, PI / 2
```

من الاشياء التي تزيد من قوة الطريقة Circle هي امكانية ايصال اطراف الاقواس بمركز الدائرة وتلوين جزء معين كما يحدث مع المخططات Charts، ولعمل ذلك يشترط استخدام القيم السالبة:

```
Const PI = 3.14
FillStyle = 0
FillColor = vbBlue
Circle (Me.ScaleWidth / 2, Me.ScaleHeight / 2), 1000, , - (PI), - (PI / 2)
```

ولرسم القطع المكافئ Ellipse، استخدم المتغير لوضع النسبة Ratio بين القطر العمودي والافقي:

```
' القطر العمودي يعادل ضعف القطر الافقي
Circle (Me.ScaleWidth / 2, Me.ScaleHeight / 2), 1000, , , , 2
```

اما الطريقة PaintPicture فهي اقوى طرق الرسم والتي ظهرت منذ الاصدار VB5، الغرض من هذه الطريقة هو رسم صور تابعة للكائن او الخاصية Picture وتطلب منك 10 متغيرات! لا تخف وتتجنب استخدامها لكثرة المتغيرات، فالمطلوبة هي الثلاث الاولى اما الباقية فهي اختيارية، بالنسبة للمتغيرات فالأول هو كائن الصورة، والاربع

التالية تحدد بها المنطقة التي سترسم الصورة عليها، والاربعة التالية تحدد المنطقة التي تريد رسمها فعلا من الصورة الاصلية، والمتغير الاخير يحدد اسلوب رسم الصورة على الهدف، وهو يتطابق تماماً مع ما اوضحته سابقاً حول خاصية DrawMode.

تمكنك الطريقة PaintPicture من فعل اشياء كثيرة على الصور، كقلبها، عكس الوانها، تمديدها، تحريكها .... الخ، تجد في ملف الكتاب Codes.ZIP مثال يعرض لك تطبيقات عملية على الطريقة PaintPicture وهذا الجزء الاساسي منه:

```
Private Sub Form_Paint()
   Cls
   PaintPicture Picture1.Picture, 0, 0, lWidth1, _
        lHeight1, lX2, lY2, lWidth2, lHeight2, iDrawMode
End Sub
```

ومازال Print موجود

مازال Visual Basic محتفظا بسمات لغة BASIC القديمة، فالامر Print لا يزال موجود منذ منتصف الستينات حتى الاصدار VB6. ليس هذا فقط، بل مازالت الصيغ القديمة كالفواصل المنقوطة ";" والعادية مدعومة في Visual Basic:

```
Print "فاصلة", "عادية"
Print "فاصلة"; "منقوطة"
```

ملاحظة: رغم ان Print مصنف ضمن طرق الكائنات، الا انه –تقنيا- لا يعتبر طريقة. فهو حالة خاصة تعمدها مطوروا Visual Basic حتى تتزامن التوافيقية مع لغة BASIC.

نوع وحجم الخط الناتج من الامر Print، هو نفس القيم الموجودة في الخاصية Font. الدوال TextHeight و TextWidth تفيدان لمعرفة ارتفاع وعرض النص وتختلف باختلاف نوع وحجم الخط لتتمكن من اختيار الاحداثي المناسب لبدء الكتابة كما في الكود التالي الذي يكتب النص في وسط النافذة:

```
Dim sText As String
Font.Size = 20
sText = "تركي العسيري"
CurrentX = (ScaleWidth - TextWidth(sText)) / 2
CurrentY = (ScaleHeight - TextHeight(sText)) / 2
```

```
Print sText
```

اخيرا، جميع المخرجات النصية عبارة عن نقاط تتشكل في صورة حالها كحال طرق الرسم، والخاصية Image هي المسئولة عن حفظ المعلومات الكاملة لهذه المخرجات.

احداث النموذج

نافذة النموذج هي اكثر كائن من كائنات Visual Basic يحتوي على احداث، معظم احداثها تم شرحها في فقرة "الاحداث المشتركة". اما الاحداث الخاصة بها فتفجر من بداية تحميل النافذة حتى اغلاقها بهذا الترتيب:
Initialize -> Load -> Resize -> Activate -> Paint -> (Deactivate) -> QueryUnload -> Unload -> Terminate.

ملاحظة:	بالنسبة للحدث Deactivate فلا يتم تفجيره بعد الحدث Paint الا في حالة قيام المستخدم بتحديد نافذة اخرى في نفس البرنامج، واذا عاد المستخدم الى النافذة الاولى، فان السلسة السابقة تبدأ من الحدث Activate -> Paint -> ...

الحدث Initialize:

يتم تفجير هذا الحدث بمجرد استخدام كائن النموذج في اكوادك او انشاء نسخة جديدة من كائن النموذج، يقوم Visual Basic بتفجير هذا الحدث مبكرا جدا أي قبل انشاء نافذة النموذج ووضع الادوات عليها:

```
' يتم تفجير الحدث Initialize التابع لنموذج Form2
Dim X As Form2
Set X = New Form2
```

قد تستفيد من هذا الحدث لتعيين قيم ابتدائية للمتغيرات التابعة لنافذة النموذج قبل انشاء النافذة:

```
Dim sUserName As String

Private Sub Form_Initialize()
    sUserName = "مستخدم جديد"
```

```
End Sub
```

الحدث Load:

يتم تفجير الحدث Load بمجرد البدء في عملية تحميل النافذة باستخدام الدالة Load:

```
Load Form2
```

او حتى عند قيامك باستخدام احد خصائصها او استدعاء طرقها:

```
' يتم تفجير الحدث Load التابع لنموذج Form2
' قبل تعديل قيمة الخاصية Caption
Form2.Caption = "النافذة الثانية"
```

من الضروري معرفة ان الحدث Load لا يتسبب في ظهور النافذة فهو يقع عند تحميل وانشاء النافذة فقط، فلا تحاول استخدام الاوامر التابعة للواجهة كـ SetFocus او طرق الرسم .... الخ. قد تستفيد من هذا الحدث بوضع قيم ابتدائية كـ:

```
Private Sub Form_Load()
   Text1.Text = sUserName
End Sub
```

الحدث Resize:

بمجرد ان تظهر نافذة النموذج، فان الحدث Resize يتم تفجيره او كلما قام المستخديم بتحجيم النافذة وتغيير حجمها، قد تستخدم هذا الحدث بكثرة عند رغبتك في محاذاة الادوات او تغيير حجمها كلما قام المتسخدم بتغيير حجم النافذة:

```
Private Sub Form_Resize()
   ' توسيط الاداة على النافذة
   Command1.Move (Me.ScaleWidth - Command1.Width) / 2, _
        (Me.ScaleHeight - Command1.Height) / 2
End Sub
```

الحدث Activate:

يتم تفجير الحدث بمجرد ظهور النافذة –بعد الحدث Resize- او بمجرد كون النافذة هي النافذة النشطة Active Window. مع ذلك، لن يتم تفجير الحدث اذا انتقل المستخدم من برنامج آخر الى برنامجك، أي أن هذا الحدث لا يتم تفجيره إلا عند التنقل بين نوافذ برنامجك فقط. قد يفيدك هذا الحدث في تغيير محتويات النافذة –كتجديث البيانات- بمجرد قيام المستخدم بتغيير محتويات نافذة اخرى في نفس البرنامج:

```
Private Sub Form_Activate()
    Label1.Caption = Form2.Text1.Text
End Sub
```

الحدث Paint:

يتم تفجير هذا الحدث كلما دعت الحاجة الى اعادة رسم النافذة، فلو قمت بوضع النافذة س فوق النافذة ص ومن ثم تعود الى النافذة س، فان الحدث Paint له نصيب من الوقوع، كذلك عندما تخفي اجزاء من النافذة ومن ثم تظهرها سيتم تفجير الحدث. من الضروري جدا جدا اخبارك بانه في حالة كون قيمة الخاصية AutoRedraw تساوي True فان الحدث Paint لن يتم تفجيره حتى تحج البقرة على قرونها! افضل اكواد يمكنك وضعها بين سطور هذا الحدث هي اكواد الرسم، الكود التالي يرسم دائرة تغطي معظم اجزاء النافذة:

```
Private Sub Form_Paint()
    Cls
    FillStyle = 0
    Circle (ScaleWidth / 2, ScaleHeight / 2), _
        IIf(ScaleWidth < ScaleHeight, ScaleWidth, ScaleHeight) / 2, 0
End Sub
```

من المفيد ان اذكر هنا بان تغيير حجم النافذة يؤدي الى تفجير الحدث Paint في حالة ان قام المستخدم بتكبير الحجم، اما عند تصغير الحجم فان الحدث Paint لا يتم تفجيره، وذلك لانه لا توجد حاجة لاعادة رسم اجزاء من النافذة، فقد تلاحظ في الكود السابق انك اذا قمت بتصغيير حجم النافذة، فان الدائرة لن يتم اعادة رسمها، والفكرة الذكية التي قد تجبر Visual Basic لاعادة رسم الدائرة هي طريق الحدث Resize:

```
Private Sub Form_Resize()
   Form_Paint
End Sub
```

رغم ان الكود السابق صحيح، الا انه لا يخرج من اصابع مبرمج حريف، والسبب ان الحدث Paint سيتم تنفيذه مرتين كلما قام المستخدم بتكبيير حجم النافذة، فالأولى بسبب اعادة الرسم والثانية بسبب الاستدعاء الموجود في الحدث Resize، لذلك تجد ان المبرمج الذكي لا يستدعي الحدث Fomr_Paint مباشرة بل يترك الامر لـ Visual Basic ليفعله عند وقت الحاجة باستخدام الطريقة Refresh:

```
Private Sub Form_Resize()
   Me.Refresh
End Sub
```

بعد الحدث التلقائي الاخير Paint تكون نافذة النموذج جاهزة لاستقبال الاحداث الخاصة لباقي الادوات او احداثها الاخرى كـ Click وغيرها، اما في حالة عدم وجود أي اداة قابلة لاستقبال التركيز، فان الحدث GotFocus الخاص بنافذة النموذج سيتم تفجيره فورا.

الحدث Deactivate:

هو عكس الحدث Activate ويتم تفجيره بمجرد ان ينتقل التركيز الى نافذة اخرى تابعة لبرنامجك فقط. قد ينفذ هذا الحدث ايضا في حالة الاخفاء المؤقت للنافذة باستخدام الطريقة Hide او تعديل قيمة الخاصية Visible الى False.

الحدث QueryUnload:

يتم تنفيذ الحدث QueryUnload عندما تكون النافذة على وشك الازالة النهائية من الذاكرة –وليس الاخفاء المؤقت. يمكنك هذا الحدث من الاستعلام عن الطريقة التي تسببت في اغلاق النافذة عن طريق المتغير المرسل UnloadMode. المزيد ايضا، تستطيع الغاء فكرة اغلاق النافذة عن طريق اسناد القيمة True الى المتغير المرسل Cancel، فالكود التالي لن يمكن المستخدم من اغلاق النافذة باستخدام صندوق التحكم Control Box او الزر اغلاق "X" الموجود في اعلى النافذة:

```
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
   If UnloadMode = vbFormControlMenu Then
      Cancel = True
```

```
    End If
End Sub
```

طرق الاستعلام الاخرى عن قيم المتغير UnloadMode موجودة في تعليمات MSDN.

الحدث Unload:

ان لم تقم بالغاء عملية اغلاق النافذة في الحدث السابق، فان الحدث Unload هو الحدث التالي، معطيك فرصة اخيرة لالغاء عملية اغلاق النافذة عن طريق نفس المتغير المرسل Cancel، اما بالنسبة للمتغير UnloadMode فهو غير موجود.

الحدث Terminate:

يتم تفجير هذا الحدث بمجرد موت كائن النموذج، موضوع موت الكائنات هو احد فقرات الفصل الخامس "البرمجة كائنية التوجه OOP".

القوائم Menus

يمكنك Visual Basic من تصميم قائمة Menu لنوافذ النماذج وقت التصميم عن طريق صندوق الحوار محرر النماذج Menu Editor، حدد نافذة النموذج ثم اختر الامر ...Menu Editor من قائمة Tools. واذا كنت تعاني من كثرة اعادة تكرار تعبئة محتويات القوائم، تستطيع استخدام قوالب القوائب Template Menu عن طريق الاضافة Add-In مدير القوالب VB6 Template Manager.

مبدئيا، كل وحدة من وحدات القائمة تحتوي على الخاصية Caption التي تمثل النص الظاهر على القائمة، استخدم الرمز "&" لوضع خط تحت الحرف الذي يليه حتى تمكن المستخدم من الوصول الى الامر في القائمة بالضغط على المفتاح Alt والحرف الذي يلي الرمز، واذا كانت قيمة الخاصية Caption الرمز "-" فقط، فان القائمة ستكون عبارة عن خط فاصل. اما الخاصية Name تمثل الاسم البرمجي للقائمة والذي تنطبق عليه نفس شروط الادوات في التسيمة، فالقائمة ماهي الا أداة لكن من نوع خاص، فبامكانك كتابة اكواد تعدل في خصائص القائمة وقت التنفيذ:

```
mnuFile.Caption = "&ملف"
mnuEdit.Enabled = False
```

كما ان الخصائص Visible و Enabled موجودة في القـوائم وتـؤثر حتـى فـي القـوائم الفرعية التابعة لها. والخاصية Checked تحدد مـا اذا كنـت تريـد وضـع علامـة اختيـار بجانب عنـوان القائمـة. امـا الخاصـية WindowList فهـي تمكـن القائمـة مـن عـرض جميع النوافذ المحضونة في النافذة من النوع MDI.

القوائم المنبثقة Pop-Up Menus:
اذا نقرت بـزر الفـأرة الايمـن علـى أي كـائن، فـان قائمـة صـغيرة سـتظهر لـك. هـذه القائمـة تسـمى Pop-Up Menu. تسـتطيع تطبيقهـا فـي Visual Basic عـن طريـق الامر PopupMenu مع تحديد القائمة التي تود عرضها:

```
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, _
                X As Single, Y As Single)
   If Button And vbRightButton Then
      PopupMenu mnuView
   End If
End Sub
```

كما يمكنك عرض قائمة تابعة لنافذة نموذج اخرى:

```
PopupMenu frmMain.mnuHelp
```

الادوات الداخلية

فيمـا يلـي عـرض ملخـص لجميـع الادوات الداخليـة الموجـودة فـي صـندوق الادوات ToolBox والبداية مع أداة العنوان Label:

أداة العنوان Label

اداة العنوان من الادوات المعدومة النوافذ Windowless Controls حيث تعرض الـنص الموجود فـي الخاصـية Caption التابعـة لهـا، فـي حالـة كتابـة الرمـز "&" قبـل احـد الحروف في هذه الخاصية، فان خـط صـغير يـتم تسـطيره تحـت ذلـك الحـرف يمكـن المسـتخدم مـن نقـل التركيـز الـى الاداة التـي تلـي أداة العنـوان فـي الخاصـية TabIndex اذا ضـغط علـى المفتـاح [Alt] وذلـك الحـرف، تسـتطيع الغـاء الخدمـة السـابقة بتحويل قيمة الخاصية UseMnemonic الى False.

ملاحظة: اذا اردت عرض الرمز "&" على الأداة وكانت قيمة الخاصية UseMnemonic تساوي True، فيشترط كتابة الرمز مرتين.

الخاصية BorderStyle تظهر حدود حول الاداة، والخاصية Alignment تحاذي النص الموجود في الاداة اما من اليسار الى اليمين 0-Left Justify، من اليمين الى اليسار 1-Right Justify او في الوسط 2-Center. اما الخاصية WordWrap فهي مفيدة جدا للنصوص الطويلة حيث تقوم بازاحة النص الى سطر جديد كلما وصل حدود الأداة. الخاصية BackStyle تحدد ما اذا كنت تريد جعل أداة العنوان شفافة بحيث تظهر الادوات التي خلفها او لا.
بالاضافة الى عرض النصوص، يوجد استخدام جميل لاداة العنوان اطبقه بكثرة في برامجي، حيث اضع مجموعة ادوات العنوان على النافذة التي تحتوي على صورة لازرار واقوم بكتابة بعض الاكواد في الحدث Click لكل أداة، ولحبك الحيلة أقوم بوضع تلميح ToolTip لكل أداة مما يوحي للمستخدم ان الازرار الموجودة على الصورة حقيقية.

أداة النص TextBox

أداة النص Text Box من اكثر الادوات استخداما في تطبيقات Windows بشكل عام، فهي الوسيلة المثلى للتفاعل مع المستخدم والحصول على قيم المدخلات منه. بعد ان تضيف أداة نص جديدة على النافذة، امسح النص الابتدائي لها عن طريق الخاصية Text. واذا اردت منع المستخدم من تغيير محتويات أداة النص، فالقيمة True للخاصية Locked تفي بالغرض. كما ان الخاصية MaxLength تحدد العدد الاقصى من الحروف التي يمكن ان يكتبها المستخدم. تستطيع تحديد حرف معين كالنجمة "*" لتظهر بمقدار عدد الحروف المكتوبة عن طريق الخاصية PasswordChar، ومن الواضح ان الغرض الرئيس لها لكلمات السر.

ملاحظة: اذا استخدمت الخاصية PasswordChar، فان المستخدم لن يتمكن من سرقة النص المكتوب على الأداة باختيار الامر Copy من القائمة المنسدلة بعد النقر بزر الفأرة الايمن على أداة النص، لأن ذاكرة Visual Basic لا تنسى الغاء اوامر النسخ والقص من القائمة السابقة. اما لو انشأت قوائم بها اوامر نسخ ولصق، فذاكرتك هي المسؤولة عن الغاء او عدم تمكين هذه الوظائف.

استخدم الخاصية MultiLine لتمكن المستخدم من تحرير النص على عدة سطور، ولا تنسى الخاصية ScorllBars فهي تتحكم بظهور او اخفاء اشرطة التمرير.

ملاحظة:	اذا كانت قيمة الخاصية MultiLine هي True وقيمة الخاصية ScorllBars هي 0-None او 2-Vertical، فان النص الذي يكتبه المستخدم سيتم ازاحته الى سطر جديد بمجرد الوصول الى حدود الاداة –كالخاصية WordWrap لاداة العنوان.

من خصائص وقت التنفيذ Run Time Properties التابعة لاداة النص هي خصائص تمكنك من تحديد نص معين، حيث تضع نقطة البداية في الخاصية SelStart وطول التحديد في الخاصية SelLength. الكود التالي يقوم بتحديد النص بمجرد انتقال التركيز الى أداة النص:

```
Private Sub Text1_GotFocus()
   Text1.SelStart = 0
   Text1.SelLength = Len(Text1.Text)
End Sub
```

واذا اردت معرفة او استبدال النص المحدد فاستخدم الخاصية SelText. اما الخاصية Text فهي تمثل كامل النص الموجود في الاداة سواء كان محددا او لا، فلو اردت اضافة نص الى الاداة دون حذف النص الموجود بها فاكتب شيئا مثل:

```
Text1.SelText = "نص اضافي"
```

من الضروري التنويه هنا بان المستخدم لن يستطيع استخدام مفتاح الجدولة [TAB] اثناء الكتابة في خانة النص، والسبب في ذلك منطقي، فالمفتاح [TAB] يؤدي الى انتقال التركيز الى الادوات الاخرى، تستطيع اللتفاف حول هذه المشكلة البسيطة بالغاء الخاصية TabStop لجميع الادوات ومن ثم اعادتها:

```
Private Sub Text1_GotFocus()
   On Error Resume Next
   Dim ctrl As Control

   For Each ctrl In Controls
      ctrl.TabStop = False
```

```
    Next
    Err.Clear
End Sub

Private Sub Text1_LostFocus()
    On Error Resume Next
    Dim ctrl As Control

    For Each ctrl In Controls
        ctrl.TabStop = True
    Next
    Err.Clear
End Sub
```

من المشاكل التي تواجه مستخدمي Windows 2000, XP العرب هي عدم ظهور الحروف العربية بالشكل المطلوب –احيانا- عند نسخها من أداة النص والصاقها الى برنامج آخر، والسبب في ذلك يتعلق بتوزيع صفحات المحارف التابعة لترميز UNICODE لان ترميز أدوات Visual Basic ما زال مبني على جدول ASCII، لا اريد ان افصل في الموضوع اكثر من ذلك حتى لا نخرج عن مجال الفقرة، ولكنك تستطيع حل هذه المشكلة بتغيير اللغة الى اللغة العربية بالضغط على الازرار [Alt+SHIFT] او عمل ذلك برمجيا قبل عملية النسخ او القص:

```
Declare Function LoadKeyboardLayout Lib "user32" Alias _
"LoadKeyboardLayoutA" (ByVal pwszKLID As String, ByVal _
flags As Long) As Long

Sub BeforeCopyOrCut()
    LoadKeyboardLayout "00000401", 1
End Sub
```

السيطرة على المدخلات:

المشكلة التي اود ان اوضحها هو اننا حين نبرمج نتوقع ادخالات معينة من المستخدم. فمثلا، وضعت أداة نص لتجعل المستخدم يكتب عمره فبكل تأكيد ستتوقع ان يكون العمر قيمة عددية، لكن ماذا لو ادخل المستخدم حروفا؟ فانه من المؤكد أن منطق سير وسلوك تنفيذ البرنامج سيتأثر في افضل الاحوال هذا اذا لم تظهر رسالة الخطأ Run Time Error. لذلك ستضطر لكتابة اكواد اضافية لتضمن ان

أداة النص لا تحتوي الا على اعداد، ولعل الحدث المناسب لكتابة كود التحقق هو حدث KeyPress:

```
Private Sub Text1_KeyPress (KeyAscii As Integer)
  If KeyAscii < 48 Or KeyAscii > 57 Then
    ' المفتاح المدخل ليس عدد
    KeyAscii = 0
  End If
End Sub
```

مهلا مهلا اخي الكريم، عالم البرمجة لعبة عقلية ومنطقية، والامور فيه لا تتم بالسهولة التي تتوقعها! لانه ما زالت هنالك امكانية ان يدخل المستخدم حروف في أداة النص وهي باختصار: عن طريق لصق قيم حرفية من الحافظة أي بالضغط على المفتاحين Ctrl + V. فورا ستكون اجابتك الذكية جدا هي ان نكتب اكواد اضافية في حدث KeyDown نمنع فيه المستخدم من اجراء عملية اللصق. صح لسانك! لكنك نسيت طريقة اخرى للصق وهي عن طريق القائمة المختصرة التي يضيفها Visual Basic الى أداة النص والتي تظهر عن طريق زر الفأرة الايمن، والتي من خلالها يستطيع المستخدم ان يلصق النصوص!
لا توجد مشكلة الا ولها حل فهذا عالم البرمجة مشاكل وحلول. من وجهة نظري الشخصية، ارى ان افضل مكان -اقصد حدث- للتحقق من نوع قيمة النص المدخل هو الحدث Change، لكن المشكلة فيه انه يتطلب تصريح متغيرين عامين للعودة بالقيمة القديمة لأداة النص اذا كانت القيمة الجديدة ليست عددية:

```
Dim OldText As String
Dim OldSelStart As Long

Private Sub Text1_GotFocus()
' عندما يكون التركيز على الاداة
' لابد من حفظ قيمتها
  OldText = Text1.Text
  OldSelStart = Text1.SelStart
End Sub

Private Sub Text1_Change()
  If Not IsNumeric(Text1.Text) Then
' المفتاح المدخل ليس رقم
```

```
' قم باعادة عرض القيمة القديمة
     Text1.Text = OldText   ' توجد مشكلة خطيرة هنا
     Text1.SelStart = OldSelStart
   Else
' القيمة المدخلة رقمية اذا
' قم بحفظها
     OldText = Text1.Text
     OldSelStart = Text1.SelStart
   End If
End Sub
```

كما تلاحظ، في الكود السابق لك مني ضمان ان المستخدم لن يستطيع ادخال الا الارقام لكن في احد السطور كتبت التعليق "توجد مشكلة خطيرة هنا" والسبب انه عندما يتم تنفيذ السطر Text1.Text = OldText، سيقوم Visual Basic بتنفيذ الاجراء Text1_Change من جديد! أي ان هذا الاجراء سيتم تنفيذه كما يعرف في عالم البرمجة تراجعي Recursivly وهو احد اساليب الخوارزميات التراجعية Recursion. وحتى تتفادى هذه المشكلة استخدم متغير ستاتيكي يمنع حدوث ذلك:

```
Private Sub Text1_Change()
' متغير يمنع استدعاء الاجراء تراجعيا
   Static bExitNow As Boolean
   If bExitNow Then Exit Sub

   If Not IsNumeric(Text1.Text) Then
' المفتاح المدخل ليس رقم
' قم باعادة عرض القيمة القديمة
     bExitNow = True
     Text1.Text = OldText ' ذهبت المشكلة الخطيرة التي كانت هنا
     bExitNow = False
     Text1.SelStart = OldSelStart
   Else
' القيمة المدخلة رقمية اذا
' قم بحفظها
     OldText = Text1.Text
     OldSelStart = Text1.SelStart
```

```
  End If
End Sub
```

مازالت توجد مشكلة اخرى وخطيرة ايضا! وهي تتعلق بموقع المؤشر Caret الخاص بأداة النص. فالكود السابق لا يقوم بحفظ موقع المؤشر الا في حالة تغيير القيمة لأداة النص مما يتسبب في مشاكل لا نهاية لها عندما يقوم المستخدم بتغيير مكان المؤشر دون تغيير القيمة كتحريكه بلاسهم في لوحة المفاتيح او بزر الفأرة. والحل عن طريق حفظ قيمة موقع المؤشر في حالة حدوث ذلك:

```
Private Sub Text1_KeyUp (KeyCode As Integer, Shift As Integer)
  OldSelStart = Text1.SelStart
End Sub

Private Sub Text1_MouseUp (Button As Integer, Shift As Integer, _
                X As Single, Y As Single)
  OldSelStart = Text1.SelStart
End Sub

Private Sub Text1_KeyDown (KeyCode As Integer, Shift As Integer)
  OldSelStart = Text1.SelStart
End Sub

Private Sub Text1_Click()
  OldSelStart = Text1.SelStart
End Sub
```

كما يقول اخواننا الاعزاء في مصر: دي لو على جوستي، اليوزر مش حايئدر يكتب حاجه غير نمره! لكن من ناحية عقلية هل من المعقول انك ستكتب كل هذه الاكواد، كلما احتجت التحقق من ان المستخدم قام بادخال ارقام في أداة النص!؟
بكل تأكيد لو لم يكن هناك حل آخر لما كنت قد عرضت المشكلة من البداية، لان الحل يتم باختبار القيمة بعد ان ينتهي المستخدم من ادخال القيمة وليس في اثناء ذلك. ويتم ذلك عن طريق الحدث Validate الذي ظهر كحل لمبرمجي Visual Basic لمواجهة مثل هذه المشاكل بالتحديد مع اصداره السادس. حدث Validate يعمل بالتكامل مع الخاصية CausesValidation، لمعرفة كيف يتم ذلك، راقب هذا السيناريو: عندما ينتقل التركيز من أدا النص Text1 الى الاداة X، يقوم Visual Basic باختبار قيمة الخاصية CausesValidation التابعة للاداة X، واذا كانت قيمتها True،

يقوم Visual Basic بتنفيذ الحدث Validate التابع لأدا النص Text1 معطيا المبرمج فرصة لاختبار نوع القيمة. فتستطيع اختصار هذه المقالة بهذا الكود:

```
Private Sub Text1_Validate(Cancel As Boolean)
  If Not IsNumeric(Text1.Text) Then
' المفتاح المدخل ليس رقم
    Cancel = True
  End If
End Sub
```

فتلاحظ اننا قمنا بتغيير قيمة المتغير Cancel الى True حتى نلغي عملية الادخال من المستخدم ونعود بالتركيز الى أدا النص Text1. وقد يسأل سائل ويقول لماذا لا نستخدم الحدث LostFocus بدلا من Validate؟ والجواب هو ان الحدث LostFocus غير مرن! أي انك تحتاج الى اكواد اضافية لتعديل الخيارات (فلا يوجد به المتغير Cancel) المزيد ايضا، الحدث LostFocus سيقوم دائما باختبار القيمة رغم انك لا تود اختبار القيمة دائما، مثلا عندما يقوم المستخدم بالضغط على الزر الغاء الامر Cancel الخاص بصندوق الحوار الذي صممته.

زر الاوامر CommandButton

بعد اداتي العنوان والنص تبدأ باستخدام زر الاوامر CommandButton. لا يوجد الكثير لاخبرك به عنها الا الخاصيتان Default و Cancel، الاولى تضع حد اضافي حول الزر تمكن المستخدم من تنفيذ زر الاوامر بمجرد الضغط على المفتاح [ENTER] والثانية مرافقة للمفتاح [ESC]. وبالنسبة لخصائص الصور Picture، DisablePicture و DownPicture فلن تتمكن من رؤية نتائجها حتى تغير قيمة الخاصية Style الى 1-Graphical.

ملاحظة:	بالنسبة للخاصية Caption فهي تتأثر بالرمز "&" دائما كما تتأثر أداة العنوان Label بهذا الرمز عندما تكون قيمة خاصيتها UseMnemonic تساوي True.

أداة الاختيار CheckBox

تعطي هذه الاداة فرصة للمستخدم لتحديد اختيار معين اما بتفعيله او لا، قيمة التفعيل تحتجز في الخاصية Value والتي تكون اما 0-UnChecked، 1-Checked او

2-Grayed، بالنسبة للقيمة الاخيرة، فيعتقد معظم المبرمجين انها تشبه القيمة False للخاصية Enabled، وهذا الاعتقاد خاطئ 100% لان القيمة False للخاصية Enabled تمنع المستخدم من التعامل مع الاداة، بينما القيمة 2-Grayed تمكنه من ذلك، تستطيع استخدام هذه الخاصية في حالات ما بين تحديد الاختيار او لا، كتحديد مجموعة اختيارات فرعية كلها او بعضها او عدم تحديدها، في املف Codes.ZIP تجد مثالا تطبيقيا على هذه القيمة.
من الضروري ان تضع في ذهنك بان الحدث Click يتم تفجيره عند قيامك بتعديل قيمة الخاصية Value حتى لو لم يقم المستخدم بالنقر على الاداة. وبالنسبة لخصائص الصور Picture، DisablePicture و DownPicture فلن تتمكن من رؤية نتائجها حتى تغير قيمة الخاصية Style الى 1-Graphical.

ملاحظة:	بالنسبة للخاصية Caption فهي تتأثر بالرمز "&" تماماً كما تتأثر أداة زر الاوامر CommandButton بهذا الرمز.

زر الاختيار OptionButton

زر الاختيار OptionButton يسمى احيانا بزر الراديو Radio Button وهو شبيه بأداة الاختيار CheckBox، الا ان قيمة الخاصية Value تكون اما True او False. كذلك لن تستطيع جعل قيمة الخاصية Value الى True لاكثر من زر اختيار واحد في نفس المجموعة -أي نفس الاداة او نافذة النموذج الحاضنة، لذلك يفضل وضع هذه الازرار في داخل اداة الاطار Frame وترتيبها على مجموعات تناسب تصنيف الاختيارات. وبالنسبة لخصائص الصور Picture، DisablePicture و DownPicture فلن تتمكن من رؤية نتائجها حتى تغير قيمة الخاصية Style الى 1-Graphical.

ملاحظة:	بالنسبة للخاصية Caption فهي تتأثر بالرمز "&" تماماً كما تتأثر أداة زر الاوامر CommandButton بهذا الرمز.

أداة القائمة ListBox

تعرض لك هذه الاداة مجموعة من النصوص في داخل صندوق يحتوي على اشرطة تمرير ScorllBars. الخاصية Sorted تقوم بفرز محتويات الاداة فرز تصاعدي بالاستناد على حروفها الابجدية، مع العلم ان الحروف العربية ستكون –للاسف- اسفل الحروف الانجليزية. تستطيع عرض كل محتويات الاداة على شكل اعمدة وتلغى شريط التمرير العمودي لتظهر شريط التمرير الافقي، سيكون عدد الاعمدة هو

نفس العدد الموجود في الخاصية Columns رغم اني لا احبـذ هـذه الطريقـة لعـرض محتويات الاداة.
تغييـر ارتفـاع الاداة Height لـن يكـون دقيقـا كمـا هـو الحـال مـع الادوات الاخـرى، والسـبب ان Visual Basic يحاول تعديل ارتفاع الاداة حتى تعرض السـطر كـاملا فـي القائمة، فلن تستطيع عرض نصف السطر الا اذا قمت بالغـاء المحـاذاة التـي يفعلهـا Visual Basic عن طريق جعل قيمة الخاصية IntegralHeight تساوي False.
تسـتطيع تعبئة محتويات الاداة في وقت التصـميم عـن طريـق الخاصـية List او وقـت التنفيذ باستخدام الطريق AddItem:

```
List1.AddItem "الاول"
List1.AddItem "الثاني"
List1.AddItem "الثالث"
```

اذا كنت سـتضيف مئات او الاف العناصر وقت التنفيذ، فينصح باخفاء الاداة مؤقتا وبعد اضافة العناصر تعيد اظهارهامن جديد، وذلك لأن الاداة تعيد رسـم نفسـها تلقائيـاً مـع اضافة أي عنصر إليها، مما يتسبب في بطء التنفيذ وكذلك ارتعاش الاداة:

```
List1.Visible = False
For X = 0 To 10000
        List1.AddItem X
Next
List1.Visible = True
```

العناصر الجديدة تضاف الى نهاية سلسلة العناصر -اذا كانت قيمـة الخاصـية Sorted تساوي False- مالم تحدد موقعها بنفسك:

```
List1.AddItem "الثالث"
List1.AddItem "الاول", 0
List1.AddItem "الثاني", 1
```

تذكر ان تحديد موقع العنصر يؤثر في ترتيب العناصر الاخـرى. بامكانـك حـذف العنصـر باستخدام الطريقة RemoveItem او الطريقة Clear لحذف جميع العناصر:

```
List1.RemoveItem 0
List1.Clear
```

الخاصية ListIndex تعود بقيمة العنصر المحدد في الاداة، وتعود بالقيمة -1 ان لم يكن هناك أي عنصر محدد، بامكانك تعديل قيمة الخاصية ايضا، اما الخاصية Text فهي تعود بنص العنصر المحدد:

```
List1.ListIndex = 0
Print List1.Text
```

بالنسبة للخاصية ListCount فهي تعود بعدد جميع العناصر الموجودة، والتي تستخدم بكثرة مع الخاصية List التي تمكنك من الوصول الى العنصر:

```
For X = 0 To List1.ListCount
        Print List1.List (0)
Next
```

بالنسبة للخاصية MultiSelect فهي تمكن المستخدم من تحديد عدة عناصر متتالية في الاداة اذا كانت قيمتها 1-Simple او عدة عناصر غير متتالية –باستخدام المفتاح [Ctrl]- اذا كانت قيمتها 2-Extended، وتستطيع معرفة العنصر المحدد عن طريق الخاصية Selected، هذا الكود ينقل جميع العناصر المحددة من اداة القائمة الى اخرى:

```
Private Sub cmdMoveLeft_Click()
   Dim iCounter As Integer

   iCounter = 0

   Do While iCounter <= lstRight.ListCount - 1
      If lstRight.Selected(iCounter) = True Then
         lstLeft.AddItem lstRight.List(iCounter)
         lstRight.RemoveItem iCounter
         iCounter = iCounter - 1
      End If
      iCounter = iCounter + 1
   Loop
End Sub
```

تســتخدم نفــس الخاصــية Selected ايضــا لمعرفــة ادوات الاختيــار CheckBoxes المحددة بجانب اسماء العناصر والتي تظهر اذا كانت قيمـة الخاصـية Style تسـاوي 1-CheckBox.
اخيرا، الحدث Scroll يتم تفجيره بمجرد قيام المستخدم باستخدام اشـرطة التمريـر ScrollBars التابعة للاداة.

أداة القائمة ComboBox

معظـم الخصـائص والطـرق التابعـة لـلاداة ListBox السـابقة موجـودة فـي الاداة ComboBox، وذلك لان الاداة ComboBox عبارة عـن أداة ListBox قياسـية وتحتـوي على خانة نص TextBox اضافية في اعلاها، تستطيع عرض خانة النص بعـدة طـرق باسـتخدام الخاصـية Style. اذا كانـت قيمـة الخاصـية Style تسـاوي 0-Dropdown Combo فان اداة النص ستظهر مع سـهم يؤدي النقر عليه الـى ظهـور الجـزء الثـاني من الاداة –الشـبيه بـالاداة ListBox، واذا كانـت قيمـة الخاصـية تسـاوي 1-Simple Combo فكلا الجزئين سيظهران للمستخدم، اما اذا كانت قيمة الخاصية تساوي 2-Dropdown List فهي مثل القيمـة الاولـى باسـتثناء ان المسـتخدم لـن يـتمكن مـن الكتابة في خانة النص.
اما الحـديث عـن الاحـداث فهـي تحتـوي معظـم الاحـداث القياسـية والحـدث Scorll الموجود في الاداة ListBox، الا ان فريق التطوير لـ Visual Basic او –ان صح التعبيـر- المطور الذي قام ببرمجة الاداة ComboBox قد نسـي اضافة الاحداث MouseDown، MouseMove و MouseUp! اذا صادفته يوما من الايام فارجو ان تنصحه بتناول افطاره قبل الذهاب الى مكتبه في Microsoft.

أداة الصورة PictureBox

يمكنك اعتبار اداة الصـورة PictureBox علـى انهـا نافـذة نمـوذج Form بـدون شـريط عنــوان، فمعظــم خصــائص، طــرق واحــداث نافــذة النمــوذج موجــودة فــي الاداة PictureBox لذلك لا يوجد داعي لاعادة ذكرها فـي هـذه الفقـرة باسـتثناء الخاصـية AutoSize التي تعدل حجم الاداة بحيث تعرض جميع محتويات الصورة الموجودة في الاداة -أي الصـورة الموجـودة فـي الخاصـية Picture. كمـا ان الاداة PictureBox اداة حاضنة يمكنها ان تحتضن ادوات اخرى في داخلها.

أداة الصورة Image

اداة الصـورة Image هـي نسـخة مبسـطة مـن الاداة السـابقة PictureBox، فهـي تعرض الصور عن طريق خاصيتها Picture الا انها لا تدعم الخصائص المتقدمة للصـور

كـ HasDC، AutoRedraw، DrawMode .... الخ كما انها لا تدعم طرق الرسم Line، Cirlce .... الخ، والاحداث Paint، Resize .... الخ، اذا كنت لا تنوي استخدامها فقد تخسر الكثير! لان الاداة Image هي اداة من النوع معدومة النوافذ Windowless Control أي انها اسرع بكثير من الاداة PictureBox، كما ان استهلاكها لمصادر النظام System Resources اقل بكثير من الاداة PictureBox. فاذا كان استخدامك لادوات الصورة قاصرا على عرض الصور فقط، استخدم الاداة Image عوضا عن الاداة PictureBox.

اشرطة التمرير ScrollBars

تمكنك الاداتين HScorllBar و VScorllBar من محاكاة اشرطة التمرير المنتشرة في نوافذ وتطبيقات Windows. ابدأ بالخاصيتين Min و Max لتحديد مجال القيم التي يمكنك قرائتها او كتابتها عن طريق الخاصية Value، والتي تمثل الموقع الحالي للمستطيل المتحرك في شريط التمرير. بعد ذلك، حدد قيمة التغيير البسيط عن طريق الخاصية SmallChange وهي مقدار التغيير في القيمة عندما يقوم المستخدم بالنقر على احد ازرار اشرطة التمرير، اما الخاصية LargeChange فهي مقدار التغيير في القيمة عندما يقوم المستخدم بالنقر على شريط التمرير نفسه.
في لغة Delphi توجد خاصية تعرض اشرطة الادوات على نوافذ النماذج مباشرة، اما مع Visual Basic فللاسف لا توجد، اذا كنت تريد وضع اشرطة ادوات عليها، فانت المسؤول الاول والاخير عن ظهورها، اخفائها وتحريك الادوات، وبالنسبة لتحريك الادوات فيفضل احتضان الادوات في اداة PictureBox حتى تسهل عملية التحريك عليك:

```
Private Sub Form_Resize()
    If Me.ScaleHeight < picMain.Height Then
        VScroll1.Move 0, 0, VScroll1.Width, Me.ScaleHeight - HScroll1.Height
        picMain.Left = VScroll1.Width
        VScroll1.Min = 0
        VScroll1.Max = -(picMain.Height - Me.ScaleHeight)
        VScroll1.SmallChange = 50
        VScroll1.LargeChange = 500
    Else
        picMain.Left = 0
        VScroll1.Move Me.ScaleWidth + VScroll1.Width
    End If
```

```
    If Me.ScaleWidth < picMain.Width Then
        HScroll1.Move VScroll1.Width, Me.ScaleHeight - HScroll1.Height, _
                        Me.ScaleWidth - VScroll1.Width
        HScroll1.Min = 0
        HScroll1.Max = -(picMain.Width - Me.ScaleWidth)
        HScroll1.SmallChange = 50
        HScroll1.LargeChange = 500
    Else
        HScroll1.Move 0, Me.ScaleHeight + HScroll1.Height
    End If
End Sub
```

ادوات الملفات

من الواضح ان ادوات الملفات DriverListBox، DirListBox و FileListBox غرضها الاساسي هو عرض ملفات الجهاز، يعيبها انها ادوات قديمة جدا، وشكلها الخارجي لا يلائم ادوات عرض الملفات الشائعة لبيئة Windows، ورغم انني لا احبذ استخدامها، الا انني ملزم بشرحها فهي –على الاقل- تعتبر من الادوات الداخلية، وكما يقولون: "يمكنها تفك ازمة!".

في اغلب النوافذ، تعمل هذه الادوات جنبا الى جنب فتبدأ بالاداة DriverListBox وتحدد حرف محرك الاقراص الابتدائي DiskDrive عن طريق الخاصية Drive:

```
Private Sub Form_Load()
    Drive1.Drive = "C:\"
End Sub
```

ثم تنتقل الى الحدث Change حتى تغير مسار الادلة الموجودة في الاداة DirListBox في كل مرة يقوم بها المستخدم بتغيير المحرك في الاداة DriveListBox:

```
Private Sub Drive1_Change()
    Dir1.Path = Drive1.Drive
End Sub
```

من المهم جدا توقع الخطأ في الكود السابق، فلو قام المستخدم باختيار محرك اقراص لا يوجد به قرص -كالمحرك A: مثلا- فرسالة الخطأ ستنفجر في وجه المستخدم وتنهي البرنامج:

```
Private Sub Drive1_Change()
   On Error Resume Next
   Dir1.Path = Drive1.Drive
   If Err Then
      Drive1.Drive = Dir1.Path
      Err.Clear
   End If
End Sub
```

والخطوة الاخيرة تغيير محتويات الاداة FileListBox بمجرد تغيير مسار الاداة DirListBox ونكون بذلك قد ربطنا الادوات الثلاث:

```
Private Sub Dir1_Change()
   File1.Path = Dir1.Path
End Sub
```

ملاحظة: يمكنك كتابة المسار مباشرة للخاصية Path التابعة للاداة DirListBox حالها كحال الاداة FileListBox:

```
Dir1.Path = "C:\Windows"
File1.Path = "C:\Winnt”
```

بالنسبة للاداة FileListBox فهي تحتوي على خاصية مرنة تدعى Pattern التي تمكنك من تصفية الملفات وتحديد نوع معين من الامتدادات التي تظهر على الاداة:

```
File1.Pattern = “*.EXE”
File1.Pattern = “*.TXT;*.DOC;*.RTF”
File1.Pattern = “ABC*.*”
```

المزيد من عمليات التصفية تمكنك منها الخصائص المنطقية Normal، Archive، Hidden، ReadOnly و System. فلاخفاء ملفات القراءة فقط ReadOnly واظهار الملفات المخفية Hidden اكتب الكود البسيط التالي:

```
File1.ReadOnly = False
File1.Hidden = True
```

اما الخاصية FileName فهي تمثل الملف المحدد في الاداة.

ملاحظة:	ادوات الملفات الثلاثة شبيهه بأدوات القوائم ListBox و ComboBox فهي تحتوي على الخصائص List، ListCount و ListIndex ايضا.

ادوات اخرى

من الادوات الاخرى التي أود ان اختم بها هذا الفصل هي أداة الاطار Frame التي تستخدم بكثرة لحضن الادوات وتقسيمها إلى مجموعات، ويمكنك التحكم في ظهور او اخفاء الحد الخارجي لها عن طريق الخاصية BorderStyle. واداة رسم الخطوط Line التي لا يوجد داعي لذكر الشكل الذي ترسمه ولكن ما دعاني لذكره هو خصائص الموقع والحجم التابعة لها X1، X2، Y1 و Y2 فهي ليست Left، Top، Width و Height، ولكنها احداثيات نقطتي البداية والنهاية للخط الذي تريد رسمة اما اداة رسم الاشكال Shape فتستخدم لرسم شكل من بين 6 اشكال مختلفة تستطيع تحديدها عن طريق الخاصية التي تحمل نفس اسم الاداة. وأداة المؤقت Timer التي تمكنك من تنفيذ الاوامر الموجودة في حدثها الوحيد Timer كل فترة معينة، تحديد هذه الفترة تضعه في الخاصية Interval ووحدتها 0.001 ثانية. اما أداة ربط الكائنات وتضمينها OLE فهي تمكنك من استخدام تقنية OLE لوضع مستندات مختلفة من تطبيقات مختلفة في نافذة النموذج. التحدث عن تفاصيل الاداة او تقنية OLE خارج نطاق هذا الكتاب.

ملاحظة:	لا تحاول الاكثار من الاكواد الطويلة بداخل الحدث Timer للأداة Timer فذلك يسبب استهلاك كبير للمعالجة Processing من مصادر النظام System Resources مما يؤدي الى ابطاء تنفيذ برامجك والبرامج الاخرى في نظام التشغيل.

اخيرا ومع نهاية هذا الفصل، بودي عرض هذه الملاحظة التي كتبها الاستاذ سالم المالكي عند مراجعته لهذا الفصل حول التعبير "انفجار الاحداث" ☺:

ملاحظة اخيرة: قرأت في هذا الفصل من الانفجارت ما يكفي لفناء البشرية!
اقترح عليك بشدة استبداله بأحد العبارات (اطلاق/انطلاق الحدث)/(تنفيذ الحدث)/(تشغيل الحدث). ولأني اعلم أن القضية هي قناعة خاصة فإني لا استطيع أن افرض عليك رأيي. وسأحترم اختيارك.
اما إن وافقت على التغيير وكنت تريد مني المساهمة في ذلك فلا مانع لدي، من المشاركة في رفع كل هذه الالغام!!!

ووسع صدرك.

الفصل الثالث
لغة البرمجة BASIC

ان Visual Basic شخصية اصيلة معتزة بامجادها وتاريخها، فما زالت Visual Basic محتفظة بسمات لغة البرمجة BASIC التي تصنف من لغات البرمجة العليا High level programming language، لغة BASIC هي روح لغة البرمجة Visual Basic، وهي اللغة التي زادت من اسهم وشعبية Visual Basic الى جانب مصمم النماذج Form Designer. فمعظم الصيغ Syntax التي ظهرت بها اللغة منذ بداية الستينات مازالت مدعومة بشكل جيد في احدث اصدارات Visual Basic. ليس هذا فقط، بل اضيفت اليها العشرات من الدوال والصيغ البرمجية حتى تلائم قوة Visual Basic وتحاكي تطبيقات Windows في امكانياتها.

المتغيرات والثوابت

المتغيرات والثوابت هي اساس أي لغة برمجة. إن استيعاب انواع المتغيرات من المسائل الضرورية التي تمكنك من اختيار الانواع المناسبة للمتغيرات سواء لارسالها الى الدوال او لإجراء العمليات الحسابية عليها. بودي التحدث عن مبدئا قابلية الرؤية وعمر الحياة قبل الخوض في تفاصيل المتغيرات.

قابلية الرؤية وعمر الحياة

قابلية الرؤية وعمر الحياة من احد المبادئ الضرورية في جميع لغات البرمجة، و Visual Basic يعتبر لغة برمجة حقيقة تدعم هذان المبدئان.
قابلية الرؤية Visibility –او المدى Scope- للمتغير تمثل قدرة البرنامج على الوصول الى المتغير واستخدامه، فالمتغير X الموجود في الكود التالي لا يمكن الوصول اليه خارج الاجراء MySub1:

```
Sub MySub1 ()
	Dim X As Integer
	X = 20
```

```
End Sub

Sub MySub2 ()
        Print X          ` لا يمثل المتغير X السابق
End Sub
```

اما عمر الحياة LifeTime للمتغير، فهي تمثل الفترة التي يظل فيها المتغير محتفظا بقيمته، فالمتغير X الموجود في الكود السابق، سينتهي ويزال تلقائيا من الذاكرة بمجرد الخروج من الاجراء Sub1. ولكي تفهم الاسلوب الذي يتبعه Visual Basic لتطبيق مبدأ قابلية الرؤية وعمر المتغيرات، عليك معرفة انواع المتغيرات من منظور الرؤية وعمر الحياة:

المتغيرات المحلية الديناميكية:

المتغيرات المحلية الديناميكية Dynamic Local Variables هي متغيرات تولد مع السطر الذي تعلن عنها فيه داخل الاجراء وتموت بعد نهاية الاجراء مباشرة ويتم تحرير المساحة التي حجزتها هذه المتغيرات في الذاكرة، وبالنسبة لقابلية الرؤية فلن تستطيع الوصول الى هذه المتغيرات الى في نفس الاجراء الذي صرح فيه المتغير. تستخدم الكلمة المحجوزة Dim لتصريح المتغير مع كتابة اسمه ونوعه:

```
Dim sName As String
Dim iAge As Integer
```

اذا كانت الكلمة المحجوزة Option Explicit موجودة في اعلى منطقة الاعلانات العامة لنافذة النموذج او ملف البرمجة BAS، فعليك اللالتزام بالتصريح كما في الصيغة السابقة، وان لم تكن الكلمة المحجوزة Option Explicit مسطورة فيمكنك تعريف المتغير مباشرة دون الالتزام بعملية التصريح باسناد قيمة ابتدائية له:

```
sName = "تركي العسيري"
iAge = 99
```

صحيح ان الكود السابق يوفر عليك عناء تصريح المتغير لا انه غير محبذ بشكل كبير لدى المبرمجين الجادين، قد يعرض هذا المثال احد الاسباب:

```
sCompanyName = "الشركة التجارية"
Print sConpanyName  ` الناتج 0
```

الناتج من عملية الطباعة Print في الكود السابق لن يكون كما هو متوقع "الشـركة التجاريـة"، فـالمتغير المسـتخدم فـي السـطر الثـاني هـو sConpanyName ولـيس sCompanyName وهذا الخطأ كفيل في نمو الشوائب البرمجية Bugs في برامجك.
سبب آخر قد يجعلك تحبذ الالتزام بعملية التصـريح وهـو ان جميـع المتغيـرات تكـون من النوع Variant ان لم يتم تصـريح نـوع غيـر ذلـك، والنـوع Variant هـو ابطـأ انـواع المتغيرات كما سيأتي لاحقا.
في مثالنا السـابق؛ يؤدي فرض الاعلان عن المتغيرات Option Explicit إلى الاعـلان عن خطأ و توقف البرنامج. وفي جميع الحالات فإن الخطأ في كتابة اسـم المتغيـر أو اسـناد قيمة إلى متغيرات لم يتم الاعلان عنها مسبقاً سيتتسـبب في الإعـلان عـن خطأ، وسـيتوقف البرنامج ايضاً.

ملاحظة:	توفر لك بيئة التطوير المتكاملة IDE خيار يلزمك بعمليـة التصـريح أي بكتابة الكلمة المحجوزة Option Explicit في جميع وحدات برامجـك كنوافـذ النمـاذج، ملفـات البرمجـة .... الـخ. لتفعيـل الاختيـار، حـدد الاختيـار Require Variable Declaration مـن خانـة التبويـب Editor في صندوق الحوار Options.

اخيرا، القيمة الابتدائية للمتغير العددي المصرح هي 0، والحرفي يكون قيمة حرفيـة خالية ""، اما الكائنات فهي لا شـئ Nothing.

المتغيرات المحلية الستاتيكية:
قابليـة الرؤيـة للمتغيـرات المحليـة السـتاتيكية Static Local Variables هـي مثـل قابلية الرؤية للمتغيرات المحلية الديناميكية أي لن تتمكن من الوصـول اليهـا الا مـن داخل الاجراء المصرح عنها فيه، وبالنسـبة لعمر حياة المتغير الاسـتاتيكي فهو ييبقى محتفظا بقيمته حتى نهاية البرنامج اذا كـان فـي ملـف برمجـة BAS او حتـى يمـوت الكائن التابع له. لتصريح متغير سـتاتيكي اسـتخدم الكلمـة المحجـوزة Static عوضا عـن Dim:

```
Static bStaticVariable As Boolean
```

تسـتطيع جعـل جميـع المتغيـرات التابعـة للاجـراء سـتاتيكية بوضـع نفـس الكلمـة المحجوزة عند بداية الاجراء:

```
Static Sub Counter ()
	جميع المتغيرات التالية ستاتيكية '
	Dim iCounter As Integer
	Dim iCounter2 As Integer
	...
End Sub
```

لا تحاول تطبيق الكود السابق كثيرا، فالمتغيرات الستاتيكية ابطأ من المتغيرات الديناميكية الى جانب قيامها بحجز مواقع هذه المتغيرات في الذاكرة طوال فترة عمل البرنامج، فلا تحاول استخدامها الا عند الحاجة كالرغبة في تنفيذ اجراء معين لمرة واحدة مثلا او الاحتفاظ بقيمة المتغير في عداد:

```
Sub PrintData ()
	Static bIsPrinting As Boolean

	If bIsPrinting Then
		Exit Sub
	Else
		bIsPrinting = True
	End If
	...
End Sub

Sub Counter ()
	Static iCounter As Integer

	iCounter = iCounter + 1
End Sub
```

اخيرا، الكلمة المحجوزة Static لا تطبق الا على المتغيرات المحلية فلا تحاول استخدامها على متغيرات عامة او على مستوى الوحدة فهي بطبيعتها ستاتيكية.

المتغيرات على مستوى الوحدة:

لا اقصد بالوحدة اتحاد الجماعة او نادي الوحدة الرياضي، بل اقصد الوحدة البرمجية Module المتمثلة في ملف برمجة BAS او نافذة نموذج Form او فئة Class .... الخ

من الوحدات المكونة للمشروع. يمكنك تصريح متغير على مستوى الوحدة في منطقة الاعلانات العامة للوحدة أي خارج الاجراءات.
قابلية الرؤية لهذا النوع من المتغيرات يكون عام لجميع اكواد الوحدة في حالة استخدام الكلمة المحجوزة Dim او Private:

```
Dim sName As String
Dim iAge As Integer

Sub SetData ()
        sName = "تركي العسيري"
        iAge = 99
End Sub

Sub PrintData ()
        Print sName
        Print iAge
End Sub
```

اما اذا كنت تريد تعريف متغيرات عامة قابلة للوصول من جميع انحاء المشروع، فالكلمة المحجوزة Public تفي بالغرض:

```
' في ملف برمجة BAS
Public iNumberOfUsers As Integer

' في نافذة نموذج Form1
Public sCurrentUser As String

' في نافذة النموذج Form2
Private Sub Form_Load()
    If iNumberOfUsers <= 0 Then
        Exit Sub
    Else
        Me.Caption = Form1.sCurrentUser
    End If
End Sub
```

ملاحظة:	بالنسبة للكلمة المحجوزة Global فهي مازالت موجودة لضمان التوافقية مع الاصدارات القديمة لـ Visual Basic، وهي تؤدي نفس غرض الكلمة المحجوزة Public، ولكنك لن تستطيع استخدامها الا في ملفات البرمجة BAS فقط.

اما عمر الحياة لهذا النوع من المتغيرات فيكون مرافق لعمر حياة الكائن التابع له والمصرح فيه –كعمر حياة المتغيرات الستاتيكية، وبالنسبة للمتغيرات العامة المصرحة في ملفات البرمجة BAS، فستظل محتفظة بقيمتها حتى نهاية تنفيذ البرنامج.

المتغيرات

نستطيع ان نعرف المتغيرات بمنظورين، بالمنظور الرياضي يعرف المتغير على انه مجهول س يحتوي على قيمة معينة، اما بالمنظور البرمجي –وهو الاهم- يعرف المتغير على انه قيمة تحفظ في ذاكرة الجهاز. وتختلف المساحة المحجوزة لحفظ هذه القيمة باختلاف نوع المتغير، فمتغيرمن النوع Byte لا يستهلك سوى بايت واحد من ذاكرة الحاسب، في حين أن متغير من نوع String قد يحجز مساحة تصل الى 2 جيجابايت.

وفيما يلي عرض لجميع انواع المتغيرات المدعومة من قبل Visual Basic:

المتغيرات من النوع Byte:

يستطيع هذا النوع الاحتفاظ باي قيمة صحيحة ضمن المجال العددي [0، 255] وهو اصغر انواع المتغيرات اذ لا يحتجز سوى 1 بايت. بداية المتغيرات من نوع Byte كانت منذ الاصدار VB4 وكانت معظم استخداماتها في نسخة 16bit من الاصدار VB4، اذ كانت المصفوفة من النوع Byte تستخدم كثيرا عند الاتصال باجراءات API التي تتعامل مع الحروف، اما مع الاصدارات الاحدث فلن تتمكن من الاستفادة وتطبيق الطرق القديمة على المتغيرات من النوع Byte، لان الترميز المتبع UNICODE يستهلك مساحة 2 بايت للحرف الواحد وليس 1 بايت كترميز ASCII. باختصار، لا تضع في ذهنك أي قضايا حرفية Strings عند استخدامك للمتغيرات من النوع Byte خاصة عند الغوص في اعماق اجراءات API، فيمكن قصر استخدامك لها على الاعداد الصغيرة او البيانات الثنائية مع المتغيرات من نوع Byte.

المتغيرات من النوع Integer:

اسند أي قيمة عددية صحيحة في المجال [-32,768، 32,767] للمتغيرات من النوع Integer فهـي تحجـز مسـاحة 2 بايـت. وعنـد الحـديث عـن اجـراءات API الخاصـة بالحروف، فالمصفوفة من النوع Integer هي الانسب للترميز UNICODE. بعيدا عـن اجراءات API الحرفيـة، تفيـدك المتغيـرات مـن هـذا النـوع عنـد التعامـل مـع الاعـداد الصحيحة، الا انني احبذ استخدام المتغيرات مـن النـوع Long لقـدرتها علـى احتـواء قيم اكبر بكثير من المتغيرات من النـوع Integer، كمـا انهـا النـوع القياسـي لاغلـب اجراءات API. اما في حالة المصـفوفات الكبيـرة، فـانني افضـل اسـتخدام المتغيـرات من النوع Integer لتوفير 50% من مساحة الذاكرة.

المتغيرات من النوع Long:

المتغيـرات مـن نـوع Long تسـتطيع حمـل قـيم عدديـة صـحيحة فـي المجـال [-2,147,483,648، 2,147,483,647] فهـي تحجـز مسـاحة قـدرها 4 بايـت للمتغيـر الواحد، وكما ذكرت في الفقرة السابقة اني احبذ استخدامها عوضا عـن المتغيـرات من النوع Integer، فهي تحمل قيم كبيرة جدا مقللة الخـوف مـن ظهـور خطـأ وقـت التنفيذ Overflow، فلو كتبت كود يقرأ حجم ملف معين وكنـت مـن المـدمنين للنـوع Integer، فستصاب بخيبة امل كبيرة عندما تتعامل مع الملفات التي تزيـد احجامهـا عن 32,767:

```
Dim iFileSize As Integer

' سيظهر خطأ اذا زاد حجم الملف عن 32,676 بايت
iFileSize = FileLen ("C:\MyFile.DAT")
```

المتغيرات من النوع Boolean:

المتغيرات من النوع Boolean هي نفس المتغيرات مـن النـوع Integer ولكـن القـيم التي تمكنك من اسنادها اليها تكون اما False 0 او -1 True، حجـم المتغيـرات مـن النـوع Boolean مثـل حجـم المتغيـرات مـن النـواع Integer أي 2 بايـت، الا انهـا لا تستخدم سوى 1 بت متجاهلة الـ 15 بت الاخرى. صـحيح ان الحجـم 2 بايـت يعتبـر زيادة غير مستخدمة، الا ان المتغيرات من النوع Boolean تسـهل عليك عملية قراءة وفهم الاكواد.

المتغيرات من النوع Single:

مجال القيم التي يمكن للمتغيرات مـن النـوع Single احتوائها هـو الاعـداد الموجبـة من 1.401298e-45 الى 3.402823e38 او الاعداد السالبة من 3.402823e38- الى 1.401298e-45- وتستهلك مساحة 4 بايت.

ربمـا يفضـل معظـم مبرمجـي Visual Basic النـوع Single علـى النـوع Double لاعتقادهم ان الأول اسرع في التنفيذ من الثاني، هـذا الاعتقـاد صـحيح فـي النـوع القديم من المعالجات والتي لا تحتوي علـى مسـاعد رياضـي Math Coprocessor، اما اغلب المعالجات الجديد تحتوي على المساعد الرياضـي وهـو خـاص بالعمليـات الحسابية للاعداد ذات الفاصلة العائمة Floating Point مما يجعل السـرعة متقاربـة جدا بين النوعين Single و Double، لذلك ينصح باستخدام النوع Double عوضا عـن النوع Single حتى تتقي شر الخطأ OverFlow ودقة اعلى للاعداد لكبر مجال القيم الممكنة بها. من ناحية اخرى، قد تكون المتغيرات من النوع Single اسرع بكثير مـن المتغيرات من النوع Double عند التعامل مـع الخصـائص او الطـرق التـي تحتـك مـع الاحداثيات بشـكل ملحـوظ كــ ScaleHeight، ScaleWidth، Line، Circle، CurrentX .... الخ فهذه الاحداثيات تستخدم النوع Single، واستخدام النوع Double معها ابطأ لان Visual Basic يضطر الى تحويل متغيرات النوع السابق الى Single.

المتغيرات من النوع Double:

مجال القيم التي يمكن للمتغيرات من النوع Double احتوائها هـو الاعـداد الموجبـة من 4.9406564581247e-324 الى 1.79769313486232e308 او الاعـداد السـالبة مــن 4.9406564581247e-324- الــى 1.79769313486232e308- وتســتهلك مساحة 8 بايت.

معظم دوال Visual Basic الخاصة بالاعداد تعود بقيمة مـن النـوع Double لـذلك هـو النوع المفضل دائما، الا ان عيبه الوحيد هو في المساحة الكبير التي يحتجزها، وقد يظهر هذا العيب جليا في المصفوفات الكبيرة من النوع Double.

المتغيرات من النوع Currency:

يمكن للمتغيرات من النوع Currency الاحتفاظ بقيم عشرية للفاصلة الثابتـة Fixed-Point شريطة ان تكون محصورة في داخل المجال [922,337,203,685,477.5808-، 922,337,203,685,477.5808] وحجمها 8 بايت ايضا. يوفر هذا النوع من المتغيـرات عناء التقريـب باسـتخدام دوال التقريـب كــ Fix، Round .... الـخ والتـي تسـتخدمها بكثرة مع المتغيرات من النوع Double و Single مما يبطئ العمليات الحسـابية، مـع ذلك الاستخدام المجرد للمتغيرات من النوع Currency ابطأ خمس او اربع مرات مـن

المتغيرات Double و Single فلا تستخدمها بكثرة في حالة تطبيق آلاف العمليات الحسابية عليها.

المتغيرات من النوع Decimal:

الاعداد التي يمكنك اسنادها الى المتغيرات من النوع Decimal كبيرة جدا، ولا يوجد داعي لذكرها هنا مادامت مكتبة MSDN على قيد الحياة. لن تستطيع تصريح المتغيرات من النوع Decimal مباشرة بالطريقة التقليدية Dim X As Decimal، وانما تستخدم النوع Variant -الذي يستهلك 16 بايت- ومن ثم تسند قيمة له:

```
Dim X As Variant
X = CDec (Text1.Text) * CDec (Text2.Text)
```

ولا تنسى ان المتغيرات من النوع Variant هي ابطأ انواع المتغيرات كما ستقرأ في فقرة "المتغيرات من النوع Varaint" قريبا.

المتغيرات من النوع Date:

هذا النوع من المتغيرات يحمل قيم تاريخية تبدأ من التاريخ 1 يناير 100 الى 31 ديسمبر 9999 ويشمل نفس المتغير وقت يبدأ من الساعة 00:00:00 ص حتى الساعة 23:59:59 م وتستهلك مساحة 8 بايت، وفي حقيقة الامر المتغيرات من النوع Date هي نفس المتغيرات من النوع Double، فالجزء العشري يمثل وقت معين والجزء الصحيح يمثل تاريخ معين، فالقيمة 37257.5 تمثل الساعة الثانية عشر ظهرا من يوم 1 يناير عام 2002. السبب الذي جعلني اذكر تفاصيل المتغيرات من هذا النوع هو اعطائك افكار مرنة تمكنك من اجراء عمليات كثيرة على قيم التاريخ وهذه قطرات من محيط الامثلة:

```
Dim dDateVar As Date

dDateVar = Now
' اطبع التاريخ فقط
Print Int(dDateVar)
' اطبع الوقت فقط
Print CDate(dDateVar - Int(dDateVar))
' اضف اسبوع واحد
Print dDateVar + 7
' احذف 30 يوم
```

```
Print dDateVar - 30
احذف 6 ساعات '
Print dDateVar - 0.75
```

ملاحظة:	ان لم تكن دقيق في كتابة الاعـداد المناسـبة –وخصوصـا العشـرية، فان نتائج العمليات السابقة لن تكون متوقعة، لذلك ينصح باسـتخدام دوال الوقت والتاريخ المضمنة في مكتبات VB و VBA كما سيفصـلها لك الفصل القادم.

المتغيرات من النوع String:

لماذا لغة الـ BASIC سهلة؟ والجواب بسبب المتغيرات الحرفية مـن نـوع String! اذا كنت من مبرمجي C فانسـى كل شئ متعلق بقضـية حجـز المسـاحة فـي الـذاكرة سواء كان ديناميكيا او ستاتيكيا باستخدام المصفوفات، او التحقـق مـن طـول الـنص وغيرها من الامور التي تتطلب 3 او 6 سـطور لاسـناد قيمـة الـى متغيـر حرفـي، فــ Visual Basic هو المتكفل بهذه الامور تلقائيا بمجرد تصريح متغيـر مـن النـوع String او اسناد قيم حرفية له.

منذ الاصدار VB4 –نسخة عيار 32بـت- اصـبحت المتغيـرات الحرفيـة Strings تعتمـد ترميـز UNICODE ولـيس ASCII. بصـفة عامـة، يوجـد نوعـان مـن انـواع المتغيـرات الحرفيـة يوفرهمـا Visual Basic لـك همـا المتغيـرات ثابتـة الطـول Fixed-length والمتغيرة الطول Variable-Length.

المتغيرات ثابتة الطول هي متغيرات حرفية عدد حروفها محدد في اثناء تصـريحها ولا يمكن ان يتغير:

```
Dim FixedStr As String * 12

sFixedStr = "تركي العسيري"
```

فالعدد الاقصى مـن الحـروف التـي يمكـن للمتغيـر FixedStr ان يحملـه هـو 12 ممـا يؤدي الى استهلاك مساحة قدرها 24 بايـت –لا تنسـى ان UNICODE يسـتهلك 2 بايت للحرف الواحد. من عيوب المتغيـرات ثابتـة الطـول هـو عـدم توافقهـا مـع تقنيـة COM ومعظم دوال مكتبات VB و VBA الداخلية واجراءات API لا تدعم هذا النوع من المتغيـرات، وحتـى لـو كـان عـدد حـروف القيمـة المسـندة اقـل مـن عـدد الحـروف المصرحة، فان المسافات " " ستحل محل الخانات الفارغة، ولا يمكن لهذا النوع من المتغيرات ان تكون مرئيـة علـى مسـتوى الوحـدة مـن النـوع Public، كمـا لا يمكنـه حمل عدد من الحروف اكبر من 64 كيلوبايت، الا ان الميزة التي تظهر بها عند اسناد

القيم الحرفية لهذا المتغيرات فنتائجها تكون دائما اسرع من المتغيرات من النوع المتغيرة الطول، وذلك لان Visual Basic لا يقوم باي عمليات احتجاز في الذاكرة والتحقق من المساحة المتوفرة .... الخ.

ملاحظة:	COM او برمجة الكائنات المكونة Component Object Model من التقنيات المبنية على OLE والتي تمكن تطبيقات Windows من الاتصال وتبادل البيانات فيما بينها، الفصلان الثاني عشر والثالث عشر "برمجة المكونات COM" يختصان بهذه التقنية.

بالنسبة للمتغيرات المتغيرة الطول Variable-Length فهي باختصار تغطي على جميع عيوب النوع السابق، الا انها تحتجز مساحة تعادل ضعف عدد الحروف + 10 بايتات اضافية تحوي معلومات عن المتغير الحرفي كحجمه وغيرها من التفاصيل التي يخفيها Visual Basic عنك، والعدد الاقصى من الحروف التي يمكن حفظها في هذا النوع يصل إلى 2 جيجا بايت.

المتغيرات من النوع Object:

معظم المتغيرات التي تمثل كائنات سواء صرحت بالنوع Object او بنوع فئات هي متغيرات من النوع Object:

```
Dim X As Object
Dim Y As Form
Dim Z As Text
```

اود ان اؤجل شرح تفاصيل الكائنات –المتغيرات من النوع Object- حتى الوصول الى الفصل الخامس "البرمجة كائنية التوجه" وحتى ذلك الحين، لا تسند كائن الى كائن الا باستخدام العبارة Set:

```
Set X = New MyClass
Set Y = Form1
Set Z = Text1
Z = "اسناد قيمة خاصية وليس كائن"
Z.Text = "اسناد قيمة خاصية وليس كائن"
```

لا تشغل بالك كثيرا بالكود السابق، فالفصل الخامس قادم اليك.

المتغيرات من النوع Variant:

ظهرت المتغيرات من النـوع Variant فـي الاصـدار VB3 وتعـدلت بنيتـه التحتيـة منـذ الاصدار VB4 حتى تتوافـق مـع تقنيـة COM، ويسـتطيع حمـل جميـع انـواع البيانـات السـابق ذكرها مثل: String، Date، Long .... الخ.

الحجم الذي يسـتهلكه المتغير Variant هو 16 بايت،البايت الاول يحـدد نـوع القيمـة الموجودة في المتغير، والبايتات من 2 الى 7 لا تسـتخدم الا في حالـة كـون القيمـة من النوع Decimal، اما البايتـات مـن 8 الـى 15 فهـي تمثـل القيمـة التـي يحملهـا المتغير.

الميزة التي تتميز بها المتغيرات من نوع Variant ليس فقـط فـي امكانيـة اشـتمالها على انواع مختلفة من البيانات بل واجـراء العمليـات الحسـابية او المنطقيـة عليهـا، حيث يقوم Visual Basic باختبار نوع المتغيرات ومن ثـم اجـراء العمليـة الحسـابية او المنطقية المناسـبة لها:

```
Dim X As Variant
Dim Y As Variant
Dim Z As Variant

X = 2000              ' قيمة من النوع Integer
Y = CLng(2000)        ' قيمة من النوع Long
Z = X + Y             ' قيمة من النوع Long
X = CDbl(2.5)         ' قيمة من النوع Double
Z = X + Y             ' قيمة من النوع Double
```

لا تحاول الاعتماد على الطرق السـابقة بشـكل اسـتثنائي، فقـد تعطيـك نتـائج غيـر متوقعة، فمثلا استخدام معامل الجمع + مع متغيرين من النوع Variant يـؤدي الـى جمعهما اذا كانت قيم عددية، امـا الحرفيـة فتـتم عمليـة الـدمج بينهمـا كاسـتخدام معامل الـدمج &، واذا كـان احـد المتغيـرين حرفـي والاخـر عـددي فسـيقوم Visual Basic بمحاولة تحويل القيمة الحرفية الى عدديـة، وان لـم يسـتطع فرسـالة الخطـأ Type Mismatch سـيكون لها نصيب في الظهور:

```
Dim X As Variant
Dim Y As Variant
Dim Z As Variant

X = 20
```

```
Y = "20"
Z = X + Y      ` Z = 40
X = "20"
Z = X + Y      ` Z = "2020"
Print Z
X = 20
Y = "abcd"
Z = X + Y      ` رسالة خطأ
```

اذا فتنت في المتغيرات من النوع Variant واعجبت بها كثيرا، فتذكر انها ابطأ انواع المتغيرات، فلا تحاول الاعتماد عليها الا عند الضرورة القصوى او عند الحاجة لاستخدام المتغيرات من النوع Decimal.
تستطيع معرفة نوع القيمة الموجودة في المتغير من النوع Variant باستخدام الدالة VarType:

```
Dim X As Variant
X = 20
Print VarType(X)        ` Integer تطبع 2 وهو النوع
X = "20"
Print X                 ` String تطبع 20 وهو النوع
```

اذا لم تسند أي قيمة للمتغير Variant فان القيمة الابتدائية له هي Empty والتي تستطيع اختبارها بالدالة IsEmpty:

```
Dim X As Variant

Print IsEmpty(X)        ` True
X = "20"
Print IsEmpty(X)        ` False
X = Empty
Print IsEmpty(X)        ` True
```

اما القيمة Null فهي لا تعني Empty، لان Null لا تعتبر قيمة خالية فهي قيمة معينة تستخدم في الغالب مع قواعد البيانات DataBases:

```
Dim X As Variant

X = Null
Print IsNull(X)          ` True
Print VarType(X)         ` Null تطبع 1 وهو النوع
```

والمتغيرات من النوع Variant يمكن لها ان تحتوي كائنات Objects، لكن لا تنسى استخدام الكلمة المحجوزة Set عند اسناد قيمة كائن الى متغير، واذا اردت الاستعلام عن نوع المتغير، فلا تستخدم VarType فهي تعطي نوع قيمة الخاصية الافتراضية للكائن، اما الدالة IsObject فهي تفي بالغرض المطلوب:

```
Dim X As Variant

Set X = Text1
Print IsObject(X)        ` True
X.Text = "النص"          ` Text1.Text
```

اخيرا، المتغيرات من النوع Variant يمكن لها ان تحوي مصفوفات كما سيأتي في فقرة "التركيبات والمصفوفات"، وبالنسبة للتركيبات من النوع UDT فيمكن احتضانها في المتغيرات Variant شريطة ان يكون التركيب من النوع Public مصرح على مستوى الوحدة البرمجية Module، أو على مستوى وحدة الفئات العامة Public Classes.

الثوابت

ابسط انواع الثوابت هي الثوابت العددية والتي يمكنك كتابتها مباشرة بالنظام العشري Decimal او باضافة البادئة &H للنظام الستعشري Hexadecimal او البادئة &O للنظام الثماني:

```
` جميع الاعداد التالية تساوي 15
Print 15
Print &HF
Print &O17
```

من الضروري ان انبه هنا بان جميع الاعداد المستخدمة في النظام الستعشري Hexadecimal كـ 0، 1، 2 ....، E، F والنظام الثماني Octal والتي تكتبها في اكوادك

تعتبر في نظر Visual Basic اعداد من النوع Integer مالم تضيف الرمز & بعد نهاية العدد فسيكون من النوع Long، قد تكون جملتي السابقة ليست ذات اهمية كبيرة عند معظم المبرمجين المبتدئين، لذلك علي ان اشد انتباههم بهذا المثال:

```
' ستعشري
' &HF000 = 61440
Print &HF000        ' هنا تطبع 4096-
Print &HF000&       ' هنا تطبع 61440
'ثماني
'&O170000 = 61440
Print &O170000      'هنا تطبع 4096-
Print &O170000&     'هنا تطبع 61440
```

بعد الثوابت العددية تأتي الثوابت الحرفية Strings، والتي يشترط كتابتها بين علامتي التنصيص المزدوجة " و "، ولاستخدام علامة التنصيص " في نفس الثابت الحرفي، كررها مرتين:

```
' مخرجات الكود التالي هي:
' ثابت حرفي
' 123"456
' "
Print "ثابت حرفي"
Print "123""456"
Print """"
```

فكرة الثوابت المسماة شبيهه بفكرة المتغيرات، ويكمن الفرق بينهما في أن قيم الثوابت لايمكنك تعديلها وقت التنفيذ لانها قيم ليست موجودة بالذاكرة كقيم المتغيرات، وانما يتم استبدال هذه الاسماء بقيمتها الفعلية في الكود اثناء عملية الترجمة Compiling، فالثوابت تحفظ مباشرة في الملف التنفيذي EXE للبرنامج.
تستطيع تعريف ثابت جديد باستخدام العبارة Const:

```
Const PI = 3.14

Print PI
```

كما يفضل تعريف نوع الثابت لزيادة سرعة التعامل معه:

```
Const PI As Double = 3.14
Const PROGRAMMER_NAME As String = "تركي العسيري"
Const SPECIAL_VALUE As Long = &H32FE&
```

ارجو ان تلتزم بالقيمة المناسبة عند تحديد نوع الثابت، فلا تسند قيمة عشرية لثابت صحيح -كالنوع Integer مثلا، لان قيمة الثابت ستتغير ان لم تظهر رسالة الخطأ Type Mismatch:

```
Const PI As Integer = 3.14   ' القيمة ستكون 3
Const PI As Integer = "abc"  ' ستظهر رسالة خطأ
```

اخيرا، قابلية الرؤية الافتراضية للثوابت تكون Private على مستوى الاجراء المحلي، او على مستوى نافذة النموذج او الفئة اذا صرح عنها في منطقة الاعلانات العامة، او على مستوى المشروع اذا صرح عنها في ملفات البرمجة BAS. مع تضمين الكلمة المحجوزة Public:

```
Public Const PI As Double = 3.14
```

التركيبات والمصفوفات

بالاضافة الى انواع المتغيرات السابقة، تستطيع تخصيص انواع جديدة تعرف بالتركيبات، كما يمكنك ربط سلسلة من المتغيرات في مصفوفات احادية او متعددة الابعاد.

تركيبات Enum

يمكنك تعريف نوع جديد من المتغيرات بحيث يحتوي على قيمة من مجموعة قيم تعرف بالـ Enumeration. تستطيع استخدام الكلمة المحجوزة Enum لتعريف التركيب شريطة ان يكون في منطقة الاعلانات العامة، هذا مثال يعرف تركيب لأيام الاسبوع:

```
Private Enum enmDay
    Saturday
    SunDay
    MonDay
```

```
    TuesDay
    Wednessday
    Thursday
    Friday
End Enum
```

والان يمكنك استخدام التركيب السابق لتعريف انواع جديدة من المتغيرات:

```
Dim X As enmDay
Dim Y As enmDay

X = Saturday
Y = X
```

او حتى استخدامها لاستقبال المتغيرات في اعلى الاجراءات:

```
Private Sub MySub(TheDay As enmDay)
    If TheDay = Friday Then
        MsgBox "اجازة"
        Exit Sub
    End If
End Sub
```

حقيقة المتغيرات من النوع تركيبات Enum ماهي الا متغيرات عددية من النوع Long فتستطيع التعامل معها كما لو كانت متغيرات عددية:

```
Dim X As enmDay

X = Saturday
Print X
X = X + 1
Print X
```

كما تلاحظ، يبدأ ترقيم عناصر التركيب من العدد 0، ولتخصيص قيم من عندك، ارجع الى تعريف التركيب وضع القيم من عندك:

```
Private Enum enmDay
    Saturday = 20
    SunDay = 30
    MonDay
    TuesDay
    Wednessday
    Thursday
    Friday
End Enum
```

مع العلم ان مقدار الزيادة لباقي العناصر تكون 1.

تركيبات من النوع UDT

يعرف هذا النوع من التركيبات بالانواع المعرفة من قبـل المسـتخدم User Defined Types حيث تمكنك هذه التركيبات من الاحتـواء علـى انـواع مختلفـة مـن البيانـات، استخدم الكلمة المحجوزة Type لتعريف تركيب جديد:

```
Private Type typPerson
    sName As String
    bSingle As Boolean
    iAge As Integer
End Type
```

ويمكنك استخدامه مباشرة كما في هذا الكود:

```
Dim Turki As typPerson
Dim Ali As typPerson

Turki.sName = "تركي العسيري"
Turki.iAge = 99
Turki.bSingle = True

Ali.sName = "علي العلي"
Ali.iAge = 35
Ali.bSingle = False
```

بل يمكنك نسخ كآفة قيم التركيب الى تركيب آخر من نفس النوع:

```
Ali = Turki
Print Ali.sName
```

ولمعرفة حجم التركيب، فالدالة LenB تفي بالغرض:

```
Print LenB (Turki)
```

لا تنسى انه يمكن للتركيبات ان تحتوي على تركيبات اخرى:

```
Private Type typAdress
    sCountrey As String
    sCity As String
End Type

Private Type typPerson
    sName As String
    bSingle As Boolean
    iAge As Integer
    Address As typAdress
End Type
```

الوصول الى عناصر التركيب المحضن يتم من خلال التركيب الحاضن لها:

```
Dim Turki As typPerson

Turki.sName = "تركي العسيري"
Turki.iAge = 99
Turki.bSingle = True
Turki.Address.sCity = "الظهران"
Turki.Address.sCountrey = "المملكة العربية السعودية"
```

بالنسبة لقابلية الرؤية فلن تستطيع تعريف التركيبات باستخدام الكلمة المحجوزة Public الا في الفئات Classes، واذا كنت عنيدا واصررت على استخدام الكلمة المحجوزة Public في ملفات البرمجة BAS، فسيسمح لك Visual Basic بتصريح

متغيرات جديدة من التركيب العام، ولكنك ستصاب بخيبة امل كبيرة ان علمت ان الاجراءت المصرحة على مستوى الوحدة Public لا يمكنك استخدام هذه التركيبات كقيم مستقبلة Parameters لها:

```
Public Sub MySub(P As typPerson)   ' غير ممكن
Private Sub MySub(P As typPerson)  ' ممكن
Freind Sub MySub(P As typPerson)   ' ممكن
```

ملاحظة:	بالنسبة للفئات Classes، تستطيع تعريف التركيبات على مستوى Public شريطة ان تكون قيمة الخاصية Instancing لا تساوي 1- Private.

المصفوفات

يمكنك Visual Basic من انشاء والتعامل مع المصفوفات Arrays سواء كانت احادية البعد او متعددة الابعاد –قد تصل الى 60 بعدا:

```
Dim OneDim (99) As Intger          ' 100 عنصر
Dim TwoDim (4, 9) As Integer       ' ثنائية الابعاد
Dim ThreeDim (2, 2, 2) As Integer  ' ثلاثية الابعاد

Dim OneDArray(0 To 10) As String
Dim TwoDArray(0 To 10, 0 To 10) As Long
Dim OneDArray(15 To 22) As String
```

تستطيع البدء في عملية اسناد القيم بمجرد تصريح المصفوفة مع العلم ان فهرس المصفوفة Array Index يبدأ من صفر مالم تستخدم الكلمة المحجوزة Option Base 1 في منطقة الاعلانات العامة للوحدة البرمجية فانه سيبدأ بواحد:

```
OneDim (0) = 100
OneDim (1) = 200
TwoDim (0, 0) = (100, OneDim (0) + OneDim (1))
```

ملاحظة:	رغم ان بدء ترقيم فهرس المصفوفة يمكن ان يبدأ بواحد، الا انني لا احبذ لمبرمجي Visual Basic فعل ذلك، فعند نقل الاكواد بين المشاريع المختلفة او الوحدات البرمجية المختلفة قد لا يتم تفعيل الكلمة المحجوزة Option Base 1 مما يترتب عنه ظهور عشرات الاخطاء البرمجية.

ولمعرفة رقم العنصر الاول استخدم الدالة LBound بينما الدالة UBound تعود برقم العنصر الاخير:

```
Dim lCounter As Long

For lCounter = LBound (OneDim) To UBound (OneDim)
        Print OneDim (lCounter)
Next
```

اما مع المصفوفات المتعددة الابعاد، عليك ارسال رقم البعد مع الدالتين UBound و LBound:

```
Print UBound (TwoDim)        ` تطبع 4
Print UBound (TwoDim, 1)     ` تطبع 4
Print UBound (TwoDim, 2)     ` تطبع 9
```

المصفوفات السابقة OneDim، TwoDim و ThreeDim هي مصفوفات ستاتيكية أي ثابتة الحجم لا تتغير في وقت التنفيذ، لذلك فالمرونة الحقيقة ستكون مع المصفوفات الديناميكية Dynamic Arrays التي تتيح لك التحكم في حجم المصفوفات كلما دعت الحاجة، وتصريحها يكون بدون ذكر حجمها:

```
Dim DynamicArray () As String
```

قبل ان تبدأ في عملية اسناد القيم، عليك استخدام الكلمة المحجوزة ReDim اولا مع ذكر الحجم:

```
ReDim DynamicArray (2)

DynamicArray (0) = "نوره"
```

```
DynamicArray (1) = "العنود"
DynamicArray (2) = "الهنوف"
```

لو اردت زيادة او تقليص حجم المصفوفة، استخدم ReDim مرة اخرى وعليك معرفة ان جميع محتويات المصفوفة سوف تلغى:

```
ReDim DynamicArray (4)

DynamicArray (3) = "جنان"
DynamicArray (4) = "زغلول!"

Print DynamicArray (4)        ' "زغلول!" تطبع
Print DynamicArray (2)        ' لا تطبع شيئ
```

واذا رغبت بتغيير حجم المصفوفة دون المخاطرة بفقد البيانات الموجودة فيها، فالكلمة المحجوزة Preserve جاهزة للاستخدام:

```
ReDim Preserve DynamicArray (4)

DynamicArray (3) = "جنان"
DynamicArray (4) = "زغلول!"

Print DynamicArray (4)        ' "زغلول!" تطبع
Print DynamicArray (2)        ' "الهنوف" تطبع
```

الحديث عن Preserve يقودني لاخبارك انك لن تستطيع تغيير ابعاد المصفوفة، فالمصفوفات الديناميكية التالية:

```
Dim OneDim () As Integer
Dim TwoDim () As Integer

ReDim OneDim (4)
ReDim TwoDim (2, 2)
```

لن تستطيع تغيير ابعادها باستخدام Preserve:

```
' مستحيل
ReDim Preserve OneDim (3, 3)
ReDim Preserve TwoDim (1)
' ولكن هذا ممكن
ReDim OneDim (3, 3)
ReDim TwoDim (1)
```

من المزايا التي اضيفت الى الاصدار VB6 للمصفوفات الديناميكية هي امكانية نسخ قيم مصفوفة كاملة الى اخرى في سطر واحد شريطة ان تكونا من نفس النوع، فبامكانك كتابة شيئا مثل:

```
Dim MyArray (20) As Integer
Dim YourArray () As Integer
MyArray (0) = 10
MyArray (1) = 20
...
YourArray () = MyArray ()
Print YourArray (0)    ' = 10
```

نقطة اخيرة حول المصفوفات الديناميكة وهي امكانية تدميرها باستخدام العبارة Erase:

```
Erase OneDim
```

النوع Variant مرة اخرى:

المتغيرات من النوع Variant يمكن لها ان تمثل مصفوفات اما عن طريق اسناد مصفوفة لها:

```
Dim Cities(2) As String
Dim vCities As Variant

Cities(0) = "الرياض"
Cities(1) = "جدة"
Cities(2) = "ابها"
vCities = Cities
Print vCities(1)          ' جدة
```

او باستخدام الدالة Array:

```
Dim vCities As Variant

vCities = Array("الرياض", " جدة", " ابها")
Print vCities(0)          ' الرياض
```

من الاشياء التي تعجبني في المتغيرات من النوع Variant هو انشاء مصفوفات مختلفة الابعاد Variable-Dimension Arrays، وهي مصفوفات ابعادها تختلف من عنصر لاخر، فقد يكون العنصر الاول احادي البعد والثاني ثنائي البعد والثالث ثلاثي البعد وتطبيقها سهل جدا، الطبخة كلها احتواء مصفوفة من النوع Variant على عدة مصفوفات:

```
Dim VarDim(2) As Variant

VarDim(0) = "احادي البعد"
VarDim(1) = Array("ثنائي البعد1", "ثنائي البعد2")
VarDim(2) = Array("ثلاثي البعد1", "ثلاثي البعد2", "ثلاثي البعد3")

Print VarDim(0)
Print VarDim(1)(0), VarDim(1)(1)
Print VarDim(2)(0), VarDim(2)(1), VarDim(2)(2)
```

المجموعات

تلعب المصفوفات دورا حيويا في برامجك الجدية خاصة بالامور التي تتعلق بالحلقات التكرارية وغيرها. الا ان المجموعات Collections تعتبر اكثر مرونة من المصفوفات من حيث اضافة وازالة العناصر منها. الهدف من هذه الفقرة هو تعريفك بالمجموعات وطرق استخداماتها.

مزايا المجموعات:

المجموعات Collections عبارة عن كائنات مشتقة من مكتبة VBA، وظيفتها الرئيسية مثل وظيفة المصفوفات تماما الى انها تختلف عنها في النقاط التالية:

- المجموعـات لا تحتـاج الـى تحديـد حجمهـا عنـد عمليـة تصـريحها. فعنـد تصـريحك لمجموعة جديدة، تستطيع اضافة العناصر لها وقت التنفيذ ديناميكيا أي بدون تحديـد أي حجم لها ستاتيكيا.

- تستطيع اضافة العناصر الى المجموعات في أي مكان تريـده فـي المجموعـة. أي لست مضطرا لوضعها في نهاية القائمة مثل المصفوفات، فالمجموعات تعطيك حرية كبيرة في اضافة وحذف العناصر سواء كانت في بداية المجموعة او نهايتهـا او حتـى في وسطها.

- المجموعة الواحدة يمكن ان تحتوي عناصرها علـى انـواع بيانـات مختلفـة، أي قـد يكون العنصر الاول فيها String والثاني Integer بعكـس المصـفوفات التـي لابـد مـن توحيد نوع عناصرها.

- تـوفر المجموعـات طـرق اخـرى لتحديـد عناصـرها عـن طريـق مفـاتيح Key بـدون استخدام اسلوب الترقيم الذي تتبعه المصفوفات.

بعد هذه المزايا قد تحرم استخدام المصفوفات وتنتقل الـى المجموعـات الـى الابـد! لكن من المهم جدا ان تضع في ذهنك ان المجموعات ابطأ من المصفوفات باكثر من 100 مرة! لذلك، لا تستخدمها ان كانـت السـرعة تعنـي لـك الشـئ الكثيـر. المزيـد ايضا، المجموعات تستهلك مساحة اكبر بكثير من المساحة المطلوبة فـي الـذاكرة، ويعيب المجموعات ان عناصرها المضافة قابلة للقـراءة لكـن ليسـت قابلـة للتعـديل! فلتعديل قيمة عنصر من عناصر المجموعة، عليك القيام باسلوب غير مباشـر كحـذف العنصر المراد تعديله واضافته من جديد بعد التعديل.

برمجة المجموعات:

الخطوة الاولى التي تحتاجها هي انشاء كائن المجموعـة والـذي تحصـل عليـه مـن النوع -الفئة- Collection:

```
Dim MyCol As New Collection
```

ولاضافة عناصر الى المجموعة، استخدم الطريقة Add مع ارسال قيمة العنصـر لهـا. كذلك، تستطيع ارسال مفتاح Key اذا اردت حتى تستخدم كطريقـة اخـرى للوصـول الى العنصر بدون الاعتماد على رقمه:

```
MyCol.Add "amazon.com", "shopping"
MyCol.Add "hotmail.com", "mail"
MyCol.Add "yahoo.com", "Search"
```

تستطيع الوصول الى عناصر المجموعة عن طريق الطريقة Item -او حتى تجاهلا- وارسال رقم العنصر او مفتاحه:

```
Print MyCol.Item(1) ' amazon.com
Print MyCol.Item("mail") ' hotmail.com
Print MyCol("Search") ' yahoo.com
```

ولحذف العنصر استخدم الطريقة Remove:

```
MyCol.Remove 1
MyCol.Remove "mail"
```

واسرع طريقة تمكنك من حذف جميع العناصر، قم بانهاء كائن المجموعة:

```
Set MyCol = Nothing
```

لكن تذكر! عليك باعادة انشاء الكائن حتى تتمكن من اضافة عناصر جديدة:

```
' خطأ في هذا السطر
MyCol.Add "amazon.com", "shopping"

' لابد من انشاء الكائن من جديد
Set MyCol = New Collection
MyCol.Add "amazon.com", " shopping"
```

اخيرا، لمعرفة عدد العناصر في المجموعة استخدم الطريقة Count:

```
Print MyCol.Count
```

الاجراءات والدوال

يمكنـك حبيـب القلـب Visual Basic مـن تعريـف اجـراءات Sub's ودوال Functions، حيث يمكن للدوال من العودة بقيمة بعد نهاية تنفيذ الدالة، نـوع القيمـة التـي تعـود بها الدالة هو النوع الذي تكتبته في نهاية تعريف الدالة:

```
' دالة تعود بقيمة حرفية
Function GetUserName () As String
        GetUserName = "تركي العسيري"
End Sub

' دالة تعود بتركيب UDT
Function GetPersonData () As typPerson
        GetPersonData.sName = "تركي العسيري"
        GetPersonData.iAge = 99
End Function
```

الكود الموجود داخل الدالة، يتعامل مع اسم الدالة كمتغير من نفس نوع الدالة:

```
Function OddNumbers() As String
   Dim iCounter As Integer

   OddNumbers  = ""
   For iCounter = 0 To 9
      If iCounter Mod 2 <> 0 Then
         OddNumbers = OddNumbers & iCounter
      End If
   Next
End Function
```

تنتهي عمليـة تنفيـذ الاجـراء او الدالـة بمجـرد الوصـول الـى نهايتـه، تسـتطيع انهـاء تنفيذهما قبل ذلـك باسـتخدام العبـارة Exit Sub للاجـراءات والعبـارة Exit Function للدوال، هذا مثال للخوارزميات التراجعية Recursion يسـتخدم Exit Function لانهـاء عملية تنفيذ الدالة:

```
Function Factorial(iNum As Integer) As Long
    If iNum = 1 Then
        Factorial = 1
        Exit Function
    Else
        Factorial = iNum * Factorial(iNum - 1)
    End If
End Function
```

بالنسبة لقابلية الرؤية للاجراءات والدوال فالافتراضية Public، أي تستطيع استدعائها من أي مكان في المشروع:

```
' في ملف برمجة BAS
Sub MySub1 ()
        ...
End Sub

' في نافذة نموذج Form1
Public Sub MySub2 ()            ' يمكنك تجاهل الكلمة المحجوزة Public
        ...
End Sub

' في نافذة النموذج Form2
Private Sub Form_Load()
        MySub1
        Form1.MySub2
End Sub
```

اما الكلمة المحجوزة Private السابقة لاسم الاجراء او الدالة، فهي تمنع المبرمج من استدعاء الاجراء من خارج الوحدة البرمجية:

```
' في نافذة نموذج Form1
Private Sub MySub ()
        ...
End Sub
```

```
في نافذة النموذج Form2 `
Private Sub Form_Load()
        Form1.MySub             ` لن تستطيع استدعاء الاجراء لانه خاص بالنموذج Form1 فقط
End Sub
```

اما تأثير الكلمة المحجوزة Friend فهو نفس تأثير الكلمة المحجوزة Public في المشاريع القياسية Standard EXE، ويكمن الفرق بينهما في حالات بناء مشاريع من النوع ActiveX حيث ان الاجراءات من النوع Public يمكن ان يستدعيها المبرمج من خارج المشروع عن طريق الاتصال COM، اما الاجراءات من النوع Friend فلن يتمكن أي شخص من استدعائها ما دامت اكواده خارج اكواد مشروعك.

الارسال بالمرجع او القيمة

الاجراءات Sub's او Functions قابلة لاستقبال متغيرات Parameters. تحديد المتغيرات التي يحتاجها الاجراء يعتمد بشكل كبير على الهدف والوظيفة التي يقوم بها الاجراء. بصفة عامة، الاجراءات سواء كانت Sub's او Functions يمكن ان تستقبل انواع مختلفة من المتغيرات سواء كانت انواع قياسية كـ Integer او String .... الخ، او حتى انواع معرفة من قبل المستخدم كتركيبات UDT او فئات Classes. بالاضافة الى قابلية الاجراءات لاستقبال المصفوفات Arrays. عملية ارسال المتغير الى الاجراء تتطلب وضع القيم بين قوسين في حالة كون الاجراء سيعود بقيمة لمستدعي ذلك الاجراء، اما غير ذلك فلا يشترط وضع الاقواس.
افتراضيا، تستقبل الاجراءات المتغيرات المرسلة لها بالمرجع. وان رغبت في جعلها تستقبل المتغيرات بالقيمة فلابد من كتابة الكلمة ByVal عند تصريح المتغير الذي يستقبله الاجراء. اما الفرق بين عملية ارسال المتغير بالمرجع وارساله بالقيمة فهو بسيط جدا. فعملية ارسال المتغير بالمرجع Reference تقوم على اساس ارسال مؤشر للمتغير أي عنوان المتغير بالذاكرة والذي عن طريقه تستطيع تعديل قيمة المتغير من الاجراء:

```
Sub Start()
هنا نقطة البداية `
  Dim X As Integer

 '  X = 0
  Print X
  MySub X
 '  X = 5
```

```
  Print X
End Sub

Sub MySub (X As Integer)
  X = 5
End Sub
```

فكما تلاحظ في الكود السابق، الاجراء MySub توقع استقبال مرجع لمتغير من نوع Integer مما يعطيه القدرة على تغيير القيمة المرسلة له. الكثير من الحلول يوفرها لك اسلوب الاستدعاء بالمرجع لعل اسهلها اجراء يقوم بتبديل قيمتين مرسلتين له:

```
Sub Swap (X As Variant, Y As Variant)
' اجراء يستبدل قيم انواع مختلفة
' من المتغيرات المرسلة
  Dim vTemp As Variant
  vTemp = X : X = Y : Y = vTemp
End Sub
```

اما الارسال بالقيمة فهو ابطأ من الارسال بالمرجع خاصة مع الحروف Strings والسبب في ذلك ان Visual Basic سيضطر لنسخ قيمة المتغير الى مكان مؤقت في الذاكرة، كذلك لن تستطيع تعديل قيمة المتغير المرسل لك. لجعل الاجراء يستقبل متغيرات بالقيمة استخدم الكلمة المحجوزة ByVal:

```
Sub Start()
' هنا نقطة البداية
  Dim X As Integer, Y As Integer

 '  X = 0, Y = 0
  Print X, Y
  MySub X, Y
 '  X = 3, Y = 0
  Print X, Y
End Sub

Sub MySub (X As Integer, ByVal Y As Integer)
  X = 5
```

```
  Y = 10
End Sub
```

ارسال انواع اخرى

جميع الانواع القياسية للمتغيرات String، Integer، Double .... الخ يمكن ان ترسلها وتستقبلها الاجراءات، اما الانواع الاخرى كالادوات Controls، كائنات مكتبات VB و VBA كـ App، Printer، Screen .... الخ والتركيبات UDT فتحتاج لشئ من التفصيل:

ارسال متغيرات الكائنات:

بالنسبة للمشاريع القياسية Standard EXE فيمكنك ارسال متغيرات تابعة لكائنات اللغة الى الاجراءات دون أي مشاكل، اما بالنسبة للمشاريع الاخرى كـ ActiveX EXE، ActiveX OCX .... الخ فلا يمكنك عمل ذلك الا للاجراءت التابعة لمشروعك وهي الاجراءات التي تصرح بالكلمة المحجوزة Private او Friend، والسبب في ذلك يبدو بديهيا اذا علمت ان الاجراءات من النوع Public يمكن ان تستدعى من برامج اخرى عن طريق COM كما ستعرف في الفصل الثاني عشر "برمجة المكونات COM 1".

ارسال المتغيرات التركيبات UDT:

المتغيرات من نوع التركيبات UDT والتي تنشئها عن طريق الكلمة المحجوزة Type تستطيع ارسالها لكن في حالات خاصة وشروط معينة اختصرها في هذه النقاط:

- اذا كان مجال التركيب من نوع Private ومعرف في داخل وحدة برمجية كنافذة نموذج، فئة، ملف برمجة الخ.. فان الاجراءت التي داخل تلك الوحدة البرمجية يمكن لها ان تستقبل المتغيرات من نوع ذلك التركيب.

- اذا كان التركيب مجاله عام Public معرف في ملف برمجة BAS فان جميع الاجراءات التي في نفس المشروع قابلة لاستقبال ذلك التركيب شريطة ان لا تكون هذه الاجراءات من النوع Public.

- في حالة كون التركيب معرف في فئة Class في مشروع من النوع ActiveX، فان الاجراءات الموجودة في برنامجك تستطيع ان تستقبل ذلك التركيب المعرف في داخل مكون COM اذا كانت الخاصية Instancing لا تساوي 1-Private.

تخصيص المتغيرات المرسلة

تستطيع تخصيص المتغيرات المرسلة اما بجعلها اختيارية Optional او غير محدودة العدد Unlimited Parameters.

المتغيرات الاختيارية:

اذا كان عدد المتغيرات المرسلة للاجراء غير عدد المتغيرات المصرحة فيه والتي يتوقعها، فان رسالة الخطأ لها نصيب. لذلك، في حالات كثيرة جدا تريد ان تجعل اجراءك مرن بما فيه الكفاية وذلك عن طريق تعريف متغيرات اختيارية تعطي المستخدم حرية ما اذا كان يريد ان يرسلها او لا. تتم هذه العملية عن طريق الكلمة المحجوزة Optional. واذا اردت معرفة ما اذا كان المستخدم قد ارسل قيمة الى الاجراء تحقق عن طريق الدالة IsMissing، لكن احذر! دالة IsMissing تعود بقيم صحيحة في حالة كون المتغير المرسل لها من نوع Variant فقط:

```
Sub Start()
هنا نقطة البداية '

 '   MyFunction = -1
   Print MyFunction()

 '   MyFunction = 4
   Print MyFunction (2)
End Sub

Function MyFunction (Optional X As Variant) As Integer
   If IsMissing(X) Then          لم يتم ارسال قيمة للمتغير '
      MyFunction = -1
   Else                          تم ارسال قيمة '
      MyFunction = X ^ 2
   End If
End Function
```

المشكلة هنا ان المتغيرات لابد ان تكون من نوع Variant حتى نتمكن التحقق من ارسالها عن طريق الدالة IsMissing وهو ابطأ الانواع استخدام. لذلك، يبدو ان الحل الافضل هو عن طريق وضع قيمة افتراضية تستخدم في حالة لم يتم ارسال قيم للمتغيرات:

```
Sub Start()
` هنا نقطة البداية

 '  MyFunction = -1
  Print MyFunction()

 '  MyFunction = 4
  Print MyFunction (2)
End Sub

Function MyFunction (Optional X As Integer = -1) As Integer
  If X = -1 Then       ` لم يتم ارسال قيمة للمتغير
    MyFunction = -1
  Else                 ` تم ارسال قيمة للمتغير
   MyFunction = X ^ 2
  End If
End Function
```

ملاحظة:	المتغيرات الاختيارية Optional لابد من ان تكون في نهاية سلسة المتغيرات المرسلة الى الاجراء -أي من جهة اليمين.

متغيرات غير محددة العدد:

في هذه الحالة، فانك لا تحدد عددا معينا من المتغيرات التي سيستقبلها الاجراء لان القيم ستكون في مصفوفة تعرفها بنفسك عن طريق استخدام الكلمة المحجوزة ParamArray، شريطة ان يكون المتغير من نوع Variant. هذا مثال لدالة توجد مجموع القيم المرسلة لها:

```
Function Sum (ParamArray args() As Variant) As Long
  Dim iCounter As Integer

  For iCounter = 0 To Ubound(args)
    Sum = Sum + args(iCounter)
  Next iCounter
End Function
```

ولاستدعاء الدالة كل هذه الامثلة صحيحة:

```
 'Sum = 10
Print Sum (5, 5)
 'Sum = 100
Print Sum (20, 20, 20, 20, 20)
 'Sum = 1000
Print Sum (250, 250, 250, 250)
```

التحكم في سير البرنامج

90% من الاجراءات لن تكون ذات قيمة معنوية كبيرة مالم تستخدم عبارات التفرع كـ If و Select او الحلقات التكرارية كـ For ... Next او Do ... Loop لتتحكم في سير البرنامج، الفقرات التالية تشرح عبارات التفرع Branch Statements وعبارات التكرار Looping Statements.

التفرع باستخدام IF

جملة If الجميلة لا يستغني عنها اي مبرمج، ليس في Visual Basic وحسب وانما في جميع لغات البرمجة. ومما لا شك فيه تعتبر If من اكثر العبارات استخداما في البرنامج، وهي تنجز اما في سطر واحد او –المفضل- عدة سطور:

```
' في سطر واحد
If X > 0 Then Y = 0
If X > 0 Then Y = 0 Else Y = X
If X > 0 Then X = 0 : Y = 0 Else Y = X

' في عدة سطور
If X > 0 Then
        Y = 0
End If

If M > 0 Then
        T = 1
Else
        T = -1
End If

If M > 0 Then
```

```
        T = 1
ElseIf M < 0 Then
        T = -1
Else
        T = 0
End If
```

اختصار الجمل الشرطية:

اذا تحقق الشرط او اصبحت نتيجة التعبير الشرطي الذي يلي جملة If هو True، فان اكواد التي تلي عبارة If ... Then سيتم تنفيذها:

```
If Value = True Then

End If

If x <> 0 Then

End If
```

تستطيع اختصار الجمل الشرطية، فلو لاحظت في جملة If الاولى اننا اختبرنا القيمة المنطقية Value اذا ما كانت صح او خطأ. المبرمجون المتمرسون يفضلون كتابتها بهذا الشكل:

```
If Value Then

End If
```

الكود السابق صحيح والسبب في ذلك، ان Visual Basic يعتبر اي قيمة غير الصفر هي True اما الصفر فهي False في حالة اختبار الشروط مع جملة If. قد تختصر جملة الشرط الثانية ايضا كما في هذا الكود:

```
If x Then

End If
```

صحيح ان النتيجة مماثلة، لكن حاول تجنب الاختصار في حالة استخدام قيم غير منطقية حتى لا تظهر لك نتائج غير متوقعة. لان القيم المناسبة للاختصارات هي القيم المنطقية فقط وفيما عدى ذلك قد يسبب لك الكثير من الشوائب. راقب الكود التالي:

```
x = 3
y = 4
If x <> 0 And y <> 0 Then   ' الطريقة الصحيحة
If x And y Then             ' مشكلة في هذا الاختصار
```

لقد استخدمت اسلوب الاختصار ويبدو ان جملتى الشرط متماثلتين، لكن هل تصدق ان الشرطين السابقين مختلفان تماما! اذا اردت معرفة السبب قم بالرجوع الى قيمة المتغيرين x و y بنظام الاعداد الثنائي 0011 و 0100 وستعرف السبب، والذي نستنتج منه باختصار على ان معاملات الربط كـ And و Or تقوم بمقارنة القيم الثنائية للعدد بغض النظر عن نوعه.

التفرع باستخدام Select

بامكانك تطبيق مبدأ التفرع باستخدام عبارة Select:

```
Select Case iDay
        Case 1
                sDay = "السبت"
        Case 2
                sDay = "الاحد"
        …
        Case 7
                sDay = "الجمعة"
        Case Else
                sDay = "غير معرف"
End Select
```

تكمن قوة عبارة Case في امكانية تطبيق المعاملات المنطقية او تحديد مجال للقيم:

```
Select Case iAge
        Case Is <= 0
                sDesc = "لم يولد"
        Case 1 To 11
                sDesc = "طفل"
        Case 15 To 20
                sDesc = "مراهق"
        Case 21 To 50
                sDesc = "رجل"
        Case Is >= 51
                sDesc = "شايب"
End Select
```

المزيد ايضا، يمكنك تحديد مجموعة قيم:

```
Select Case sLetter
        Case "A" To "B", "a" To "b"
                sLetter = "حرف ابجدي"
        Case "0" To "9"
                sLetter = "عدد"
        Case ".", ":", " ", ";", "?"
                sLetter = "رمز"
        Case
                sLetter = "غير معروف"
End Select
```

ليس هذا فقط، بل يمكنك تحديد مجموعة جمل شرطية:

```
Select Case True
        Case x > 0, Y < 0
        ' تعادل
        ' If (X > 0) Or (Y < 0)
End Select

Select Case False
        Case x > 0, Y < 0
        ' تعادل
```

```
        ' If ( Not (X > 0) ) Or ( Not (Y < 0) )
End Select
```

الحلقات التكرارية

حدد القيمة الابتدائية والقيمة النهائية للحلقة For ... Next:

```
Dim iCounter As Integer

For iCounter = 2 To 4
        Print iCounter ' سيتكرر تنفيذ الامر ثلاث مرات
Next
```

وعليك معرفة ان مقدار الزيادة سيضاف الى متغير الحلقة حتى بعد نهايتها:

```
Dim iCounter As Integer

For iCounter = 1 To 3
        Print iCounter
Next
Print iCounter ' 3 قيمة المتغير بعد نهاية الحلقة 4 وليس
```

تستطيع التحكم في مقدار الزيادة او النقصان باستخدام Step:

```
Dim iCounter As Integer

For iCounter = 10 To 0 Step -1
        Print iCounter
Next
```

يمكنك تطبيق فكرة الحلقات المتداخلة Nested Loops كهذا الكود الذي يطبع جدول الضرب:

```
Dim A As Integer
Dim b As Integer
```

```
For A = 1 To 5
   For b = A To 5
      Print A, "x", b, "=", A * b
   Next
Next
```

بامكانك انهاء الحلقة في أي وقت تريد باستخدام العبارة Exit For:

```
Dim iCounter As Integer

For iCounter = 0 To 100
        If MsgBox("هل تريد انهاء الحلقة؟", vbYesNo ) = vbYes Then
                Exit For
        End If
...
Next
```

اما حلقة For Each فهي تطبق على كائنات المجموعات Collections:

```
Dim ctrl As Control

' محاذاة جميع الادوات الى اليسار
For Each ctrl In Controls
   ctrl.Left = 0
Next
```

او حتى المصفوفات شريطة ان يكون متغير الحلقة من النوع Variant:

```
Dim X(100) As Integer
Dim Y As Variant

' كود لاسناد قيم للمصفوفة
...

' طباعة محتوياتها
For Each Y In X
   Print Y
Next
```

اما بالنسبة للحلقة Do ... Loop فهي اكثر مرونة من الحلقة For ... Next، لانك لا تحدد عدد معين من التكرار وانما جملة شرطية باستخدام While او Until:

```
Do While MsgBox("هل تريد انهاء الحلقة؟", vbYesNo ) = vbYes
        ...
Loop

Do Until MsgBox("هل تريد انهاء الحلقة؟", vbYesNo ) = vbNo
        ...
Loop
```

ستتم عملية تنفيذ الحلقة مادامت الجملة الشريط صحيحة True في حال استخدام الكلمة المحجوزة While او False في حال استخدام الكلمة المحجوزة Until. واذا اردت تنفيذ الحلقة التكرارية مرة واحد على الاقل، ضع حمل الشرط في اسفل الحلقة:

```
Do
        ...
Loop While MsgBox("هل تريد انهاء الحلقة؟", vbYesNo ) = vbNo

Do
        ...
Loop Until MsgBox("هل تريد انهاء الحلقة؟", vbYesNo ) = vbYes
```

بامكانك وضع جملة الشرط في داخل الحلقة ايضا باستخدام عبارة If او Select، لكن لا تنسى انهاء الحلقة بالعبارة Exit Do:

```
Do
        If MsgBox("هل تريد انهاء الحلقة؟", vbYesNo ) = vbYes Then
                Exit Do
        End If
...
Loop
```

التحويل بين For ... Next و Do ... Loop:

تستطيع تحويل حلقة For ... Next الى حلقة Do ... Loop والعكس صحيح، لكن عليك الانتباه الى ان القيم التي تحددها في بداية الحلقة For ... Next تمثل عدد التكرار حتى وان تغيرت، فبالرغم من ان الحلقتين التاليتين متشابهتين:

```
A = 5
For iCounter = 1 To A
        ...
Next

iCounter = 1
Do
        ...
        iCounter = iCounter + 1
Loop Until iCounter > A
```

الا ان الاختلاف سيظهر في حال ما اذا تم تغيير قيمة المتغير A، فالحلقة For ... Next سيتم تنفيذه دائما 5 مرات حتى وان تغيرت قيمة المتغير A في داخل الحلقة، بينما تغيير القيمة يؤثر بشكل كبير على عدد مرات تكرار الحلقة Do ... Loop.

تحسين الكفاءة

بصفة عامة، فان المصطلح تحسين الكفاءة Optimization يطلق على اساليب برمجية تتبع لزيادة سرعة تنفيذ الكود او التقليل من استهلاك مصادر النظام System Resources وغيرها. اما في موضوع هذه الفقرة فسنناقش تقنيات لتحسين الكفاءة والخاصة لعملية الترجمة والتي يوفرها Visual Basic من خلال خانة التبويب Compile الموجودة في صندوق حوار خصائص المشروع Project Properties.

P-Code و Native Code

عندما تقوم بتنفيذ البرنامج -بالضغط على F5- سيقوم مفسر Visual Basic بتنفيذ سطر تلو الاخر. قبل عملية تنفيذ السطر، يقوم Visual Basic بتحويل شيفرة السطر الى شيفرة من نوع P-Code حتى يفهمها المفسر وينفذ السطر. اما

Native Code فهي تحويل الشيفرة المصدرية الى لغة الالة Machine Language يفهما الجهاز مباشرة. الميزة في الملفات التنفيذية من نوع P-Code هي حجمها الصغير نسبيا كذلك توافقيتها المطلقة مع اكواد التنفيذ داخل بيئة Visual Basic. المزيد ايضا، اكواد P-Code تكون عرضة لاحداث انهيار البرنامج بنسبة اقل بكثير من اكواد Native Code. من ناحية اخرى، اكواد P-Code ابطأ من اكواد Native Code لانها ليست اكواد Machine Language حقيقية بل هي لغة مفسر Visual Basic فقط. في حالة اختيارك لترجمة الى اكواد من نوع Native Code، فسيوفر لك Visual Basic خيارات اضافية تجدها في خانة التبويب Compile الموجودة في صندوق الحوار Project Properties:

Optimize for Fast Code:

سيحاول المترجم في هذا الاختيار بتنظيم تعليمات لغة الالة بحيث تعطي اقصى نتائج لسرعة تنفيذ الاكواد بغض النظر عن حجم الملف التنفيذي EXE .

Optimize for Small Code:

سيحاول المترجم في هذا الاختيار بتقليص حجم الملف التنفيذ اقصى ما يستطيع بغض النظر عن سرعة تنفيذ الاكواد فيه.

ملاحظة:	توجد علاقة عكسية بين الخيارين السابقين، فغالباً ما يتسبب تقليص حجم الشيفرة في تخفيض سرعة البرنامج، وفي الاتجاة الآخر، غالباً ما يتسبب تحسين سرعة تنفيذ البرنامج إلى زيادة حجم الملف.

No Optimization:

لن يقوم المترجم باي محاولات لعمليات تحسين الكفاءة Optimization للملف التنفيذي.

Favor Pentium Pro:

اذا كان البرنامج سيعمل على معالج من نوع Pentium Pro فهذا الاختيار سيزيد من سرعة تنفيذ التعليمات وخصوصا الرياضية منها.

Create Symbolic Debug Info:
سيضيف هذا الاختيار تعليمات اضافية الى الملف التنفيذي لاعطاءه امكانية التنقيح Debug باستخدام برامج تنقيح الملفات التنفيذية كبرنامج التنقيح الذي توفره بيئة Microsoft Visual C. نصيحة لك، الغ هذا الاختيار.

إعدادات Advanced Optimization
يمكنك Visual Basic من تخصيص بعض خيارات تحسين الكفاءة المتقدمة والتي تجدها في صندوق حوار Advanced Optimization:

Assume No Aliasing:
هذا الاختيار سيزيد من سرعة تنفيذ البرنامج لكن من الضروري جدا عدم تطبيق مبدأ الاستعارة. والاستعارة –بشكل مبسط- هي عملية استعارة اسم لمتغير عام وارساله الى اجراء بالمرجع ByRef:

```
Dim X

Sub MySub ( Y As Integer)
  Y = 4
End Sub

Sub AliasingSub()
  ' عملية الاستعارة
  MySub X
End Sub
```

Remove Array Bound Checks:
عدم التحقق من رقم فهرس المصفوفة Array Index مما يزيد من سرعة التعامل مع المصفوفات.

Remove Integer Overflow Checks:
عدم التحقق من القيمة المرسلة الى المتغيرات الصحيحة فيما لو كانت اكبر من المجال لم لا.

Remove Floating Point Error Checks:
مثل الاختيار السابق، لكنه خاص للاعداد من نوع الفاصلة العائمة Floating Point.

Allow Unrounded Floating Point Operations:

للحصول على دقة اكبر لاعداد الفواصل العائمة.

Remove Safe Pentium™ FDIV Checks:

سيزيد من سرعة عملية القسمة لكنه قد يؤدي الى نتائج خاطئة لمعالجات Pentium والتي تعاني من مشكلة FDIV.

اعيد واكرر، مصطلح تحسين الكفاءة Optimization يطلق على اساليب وخوارزميات برمجية تتبع لزيادة سرعة تنفيذ الكود او التقليل من استهلاك مصادر النظام System Resources وغيرها. اما في هذه الفقرة فخصصت اساليب لتحسين الكفاءة والخاصة بعملية الترجمة والتي يوفرها Visual Basic من خلال خانة التبويب Compile الموجودة في صندوق الحوار خصائص المشروع Project Properties. تذكر ان هذه الاختيارات قد تسبب مشاكل او –في احسن الاحوال- نتائج غير متوقعة فاحرص على تغييرها بشكل دقيق، ولا تقول ان تركي ما نبهني!

الفصل الرابع
مكتبات VB و VBA

يوفر لك Visual Basic مئات الاجراءات والكائنات المضمنة في مكتبات VB و VBA والتي لا غنى عنها في برامجك الجدية، صحيح انك تستطيع محاكاة معظم هذه الدوال بكتابة اكواد لانجازها، الا ان استخدام الاجراءات والدوال المشمولة في مكتبات Visual Basic يعتبر افضل بكثير من انجازها بنفسك من منظور تحسين الكفاءة Optimization، فهذه الدوال صممها مبرمجون محترفون بلغات اخرى مما يجعل تنفيذها اسرع بكثير من اكوادك المكتوبة بـ Visual Basic. يأخذك هذا الفصل في جولة مع العشرات من هذه الدوال والاجراءات والتي سأتطرق اليها باختصار باختصار، اما اذا اردت شرحا وافيا لها فمكتبة MSDN بانتظارك حتى تبحر في صفحاتها.

التعامل مع الاعداد

يوفر لك Visual Basic عشرات المعاملات Operators والدوال الخاصة بالاعداد باختلاف انواعها Byte، Integer، Long .... الخ، بالاضافة الى دوال رياضية كدوال المثلثات او الدوال الاسية.

المعاملات الرياضية

يوفر لك Visual Basic المعاملات الاربع الرئيسية +، -، * و /، وفي حالة تطبيقها على انواع مختلفة من القيم، فان القيم الابسط ستتحول مؤقتا الى الاعقد – Integer الى Long و Single الى Double، بالنسبة لمعامل القسمة / فهو يقوم بتحويل جميع القيم المتمثلة في الحدين الايمن والايسر الى النوع Double، لذلك ينصح باستخدام معامل القسمة الصحيحة \ مع المتغيرات الصحيحة Byte، Integer و Long فهو اسرع اربع مرات من المعامل /:

```
Dim X As Long, Y As Long, Z As Long
Z = X / Y
Z = X \ Y ' هذا اسرع
```

كذلك معامل الاس ^ فهو يحول جميع القيم الى النوع Double، وفي احيان كثيرة لن تحتاج الا للمتغيرات الصحيحة، لذلك ينصح باستخدام معامل الضرب عوضا عن الاس:

```
Dim X As Long, Y As Long,
Y = X ^ 3
Y = X * X * X
```

اما معامل باقي القسمة MOD فيقوم بتحويل القيم الى Long مما لا يعطي دقة في التعامل مع انواع الفاصلة العائمة الاخرى كـ Double و Single، تستطيع تطوير دالة اخرى تعود بباقي القسمة للاعداد غير Long:

```
Function ModEx (dN As Double, dD As Double) As Double
        ModEx = dN – Int(dN / dD) * dD
End Function
```

بالنسبة للمعاملات المنطقية، فيوفر Visual Basic ست معاملات منطقية هي =، >، =>، <، =< و <>. بالنسبة لمعامل المساواة = فهو ليس كمعامل اسناد القيم الى المتغيرات، فمعامل المساواة = هو المعامل الذي يطبقه Visual Basic في داخل الجمل الشرطية او حتى اذا سبقه معامل اسناد آخر، فالكود التالي:

```
Dim X As Integer, Y As Integer
X = Y = 10
```

يبين لنا ان المعامل = الثاني الموجود في السطر الثاني هو معامل مقارنة المساواة وليس اسناد القيم.

ملاحظة: تفرق معظم لغات البرمجة الاخرى بين معامل المساواة ومعامل اسناد القيم، فنجد في لغة الـ C معامل المساواة هو == ومعامل اسناد القيم هو =.

اما معاملات الربط AND، XOR و NOT فهي مدعومة ايضا لربط الجمل المنطقية ويمكنك استخدامها للاعداد حيث تؤثر على البتات Bits التي تمثل قيمة العدد بالنظام الثنائي Binary.

اسبقية المعاملات:

من المفيد ان اذكر هنا، ان من الاخطاء الشائعة التي يقع فيها اغلب المبرمجين هو نسيان ان اسبقية المعامل And اعلى من المعامل Or، فمقارنة المعامل And تتم قبل مقارنة المعامل Or حتى ولو كان المعامل Or هو السابق اي في الجهة اليسرى قبل المعامل And،ففي هذا المثال:

```
Print True Or False And False
```

للوهلة الاولى يعتقد المبرمج ان النتيجة هي False بينما النتيجة الحقيقية هي True وذلك، لان المعامل الشرطي And يتم اختباره قبل المعامل الشرطي Or. ولتجنب ذلك، استخدم الاقواس:

```
Print (True Or False) And False
```

المزيد ايضا، اسبقية المعامل Not اعلى من اسبقية المعامل And اي يتم تنفيذه دائما قبل معامل And فالعبارة :

```
Print Not True And False
```

ستقوم بتنفيذ المعامل Not على كلمة True الاولى فقط حتى تكون False And False وبعد ذلك يأتي دور المعامل And وكما هو واضح فنتيجة التعبير هي False. اما اذا اردت تنفيذ المعامل And ومن ثم عكس النتيجة فتستطيع ان تستخدم الاقواس والتي لها الأسبقية الأولى على جميع المعاملات مثل:

```
Print Not (True And False)
```

وفي هذه الحالة، سيكون الناتج النهائي هو True.

الدوال الرياضية

من الدوال الرياضية التي يوفرها لك Visual Basic دالـة القيمـة المطلقـة Abs ودالـة الجذر التربيعي Sqr والدالة الاسـية Exp، اما الدالة Sgn فهي تعود بـالقيم 1 اذا كـان العدد المرسل لها موجب، والقيمة -1 اذا كان العدد المرسـل لهـا سـالب، وصـفر اذا كان العدد المرسل صفر.
بالنسبة لدالـة اللوغـارثم Log فهـي تعـود باللوغـارثم الطبيعـي للعـدد، امـا للاعـداد الاخرى، فتسـتطيع تطوير هذه الدالة:

```
Function LogEx (dN As Double, dBase As Double ) As Double
        LogEx = Log ( dN ) / Log ( dBase )
End Function
```

الكود السـابق يذكرني بتطوير دالة الجـذر التربيعـي Sqr ايضـا، حيـث يمكـن ان تعـود بالجذر النوني للعدد:

```
Function NthSqr (iNum As Integer, iRoot As Integer) As Double
  NthSqr = iNum ^ (1 / iRoot)
End Function

' مثال الحصول على الجذر التكعيبي
' لعدد 8
Print NthSqr( 8, 3) ' تعود بالعدد 2
```

اخيرا الدوال المثلثيـة Sin، Cos، Tan و Atn التـي تعـود بالقيمـة المناسـبة اسـتنادا الـى الزاويـة المرسـلة لهـا بالراديـان، امـا بالنسـبة للـدوال المثلثيـة الاخـرى Sec، Cosec، .... الخ فيمكنك اشـتقاقها بتطبيقـات معادلاتهـا المعروفـة، هـذه واحـدة مـن عندي، والبقية عليك:

```
Function Sec (X As Double) As Double
        Sec(X) = 1 / Cos(X)
End Function
```

تنسيق الاعداد

من اقوى دوال التنسيق هي دالة Format التي توفر لك خيارات لا نهائية لتنسيق الاعداد، الحروف، الوقت والتاريخ ايضا، ساسرد لك في هذه الفقرة طرق تنسيق الاعداد فقط.
الصيغة المبسطة للدالة Format تتطلب العبارة -او القيمة- و طريقة التنسيق:

```
Format ( طريقة التنسيق ,القيمة )
```

يوجد نوعان من طرق التنسيق. النوع الاول هو التنسيقات القياسية والثاني هو التنسيقات الخاصة. التنسيقات القياسية عبارة عن قيم نحدد نوع تنسيق الارقام كـ General Number لتنسيق الرقم بشكل عام او Currency لتنسيق الرقم على شكل عملة وغيرها من القيم التي تجدها في مكتبة MSDN:

```
‘ 1234567
Print Format(1234567, "General Number")
‘ 1,234,567.00 ر.س.
Print Format(1234567, "Currency”)
‘ 1,234,567
Print Format(1234567, "Standard”)
```

اما التنسيقات الخاصة فهي تنسيقات تحددها بنفسك. والتي تستخدم علامات كـ #، %، 0 .... الخ، تجدها ايضا في مكتبة MSDN:

```
' 1, 234.57
Print Format(1234.567, "#,##.00")
' 23.4%
Print Format(0.234, "#.#%")
' 020.0
Print Format(20, "00#.00")
```

دوال اخرى

من دوال حذف الفواصل الدالتين Int و Fix، الاولى تحذف الفاصلة وتحول العدد الى عدد صحيح اقل من او يساوي العدد المرسل بينما الثانية تحذف الفاصلة فقط:

```
Print Int(1.2)  ' 1
Print Int(-1.2) ' -2
Print Fix(1.2)  ' 1
Print Fix(-1.2) ' -1
```

اما دالة التقريب Round فقد ظهرت في الاصدار VB6 التي تمكنك من تحديد عدد الارقام العشرية:

```
Print Round(2.12567, 2) ' 2.13
```

وعند الحديث عن الاعداد الستعشرية Hexadecimal والثمانية Octal فان الدالتين Hex و Oct تحول اعداد النظام العشري الى الانظمة السابقة:

```
Print Hex$(100) ' 64
Print Oct$(100) ' 144
```

وللتحويل الى النظام الثنائي Binary فعليك بكتابة الدالة Bin بنفسك:

```
Public Function Bin(iNum As Integer) As String
  Dim iCounter As Integer

  Do
    If (iNum And 2 ^ iCounter) = 2 ^ iCounter Then
      Bin = "1" & Bin
    Else
      Bin = "0" & Bin
    End If
    iCounter = iCounter + 1
  Loop Until 2 ^ iCounter > iNum
End Function
```

اما الدالة Rnd فهي تعود بقيمة عشوائية اصغر من 1 واكبر من او تساوي صفر، تستطيع تخصيص مجال معين من الاعداد باستخدام هذه المعادلة:

```
Int ((اعلى قيمة – اصغر قيمة + 1) * Rnd + اصغر قيمة)
```

فللحصول على اعداد عشوائية في المجال [-2، 4] اكتب شيئا مثل:

```
Print Int(7 * Rnd + -2)
```

اخيرا، دوال تحويل القيم الى اعداد لعل اشهرها Val، لكن المفضل استخدام دوال التحويل التي تمكنك من تحديد نوع القيمة كـ CInt للاعداد Integer و CLng للاعداد Long، CDbl للاعداد Double .... الخ.

التعامل مع الحروف

من منا لا يستخدم الحروف؟ توفر لك مكتبات VB و VBA عشرات الدوال المختصة في التعامل مع المتغيرات والثوابت الحرفية Strings. اعرض عليك في هذه الفكرة معظم هذه الدوال بالاضافة الى تطبيق فكرة البحث والاستبدال Find and Replace ولكني سأبدأ بالمعاملات الحرفية.

المعاملات الحرفية

الرمز & يمثل معامل الدمج Combine Operator للقيم الحرفية:

```
Dim sMyName As String

sMyName = "تركي"
sMyName = sMyName & " العسيري"
Print sMyName          ' "تركي العسيري"
```

اما معامل الجمع "+" فأنا لا احبذ استخدامه كثيرا، فاذا كان نوع القيم حرفية فسيتحول الى معامل الجمع "&"،واذا كانت احدى القيم عددية والثانية حرفية قابلة للتحويل الى عددية فسيكون معامل جمع، اما اذا كانت احدى القيم عددية والاخرى حرفية لايمكن تحويلها الى عددية، فان رسالة الخطأ Type Mismatch لها نصيب من الظهور:

```
Print "20" + "30"       ' "2030"
Print "20" + 30         ' 50
Print "X" + 100         ' رسالة خطأ
```

بالنســبة للمعــاملات المنطقيــة >، <>، = ... الـخ فــيمكن تطبيقهــا علــى القــيم الحرفيـة ايضـا، حيـث تكـون قيمـة الحـروف هـي المقابـل لهـا فـي جـدول ASCII او UNICODE:

```
Print "Turki" > "TURKI"        ` True
Print "Turki" < "TURKI"        ` False
Print "Turki" = "TURKI"        ` False
Print "تركي" > "احمد"          ` False
Print "تركي" < "احمد"          ` True
Print "تركي" = "احمد"          ` False
```

ملاحظة:	تســتطيع ان تجعـل Visual Basic يتجاهـل مقارنـة شــكل الحـروف الكبيـرة والصـغيرة عنـد اسـتخدام معامـل المســاواة شـريطة كتابـة الكلمـة المحجـوزة Option Compare Text فـي منطقـة الاعلانـات العامة لكل وحدة برمجية.

احيانا تود تجاهـل الدقـة التـي يفرضـها عليـك معامـل المسـاواة وتسـتخدم معامـل التشـابه Like الذي يتيح لك استعمال الحـروف التعويضـية، فيمثـل الرمـز # أي رقـم والرمز ؟ أي حرف، والرمز * أي عدد معين من الحروف والارقام:

```
Dim sMyString As String

sMyString = …

If sMyString Like "A????" Then …     ` "Abcde" او "A1234"
If sMyString Like "A*" Then …        ` "Ab" او "Aabce1234"
If sMyString Like "A####" Then …  ` "A0000" او "A1234"
```

او بامكانك تحديد حروف معينة او مجال معين باستخدام الاقواس [و]:

```
Dim sMyString As String

sMyString = …

If sMyString Like "[AB]###" Then …` "A1234" او "B1234"
```

```
If sMyString Like "[AB][XY]" Then … ' "AX" او "BY"
If sMyString Like "[A-D]#" Then …   ' "C9" او "D3"
```

وحتى يمكنك استثناء حروف معينة او مجال معين باستخدام الرمز !:

```
Dim sMyString As String

sMyString = …

If sMyString Like "[!0-9]###" Then …        ' "A1234" او "Z1234"
```

البحث والاستبدال

تستطيع البحث عن كلمة او حروف معينة داخل قيمة حرفية عن طريق الدالة InStr التي تعود بموقع ذلك الحرف او بداية الكلمة:

```
Dim lPosition As Long
Dim lStartPoint As Long

lStartPoint = 1
lPosition = InStr (lStartPoint, Text1.Text, "تركي")
If lPosition > 0 Then
   Text1.SelStart = lPosition - 1
   Text1.SelLength = 4
End If
```

اما الدالة InStrRev فهي شبيهه بالدالة السابقة ولكن عملية البحث تكون معاكسة -أي تبدأ من نهاية القيمة المرسلة.
بالنسبة لعملية استبدال النصوص، فلن تجد اسرع من الدالة Replace التي تمكنك من استبدال حروف معينة بحروف اخرى. هنا سنستبدل جميع كلمات "محمد" الى "محمد صلى الله عليه وسلم" الموجودة في أداة النص:

```
Text1.Text = Replace(Text1.Text, "محمد", "محمد صلى الله عليه وسلم")
```

تنسيق الحروف

ستستخدم الدالة Format ايضا لتنسيق الحروف، ولكن لا توجد بها تنسيقات قياسية للحروف، اما التنسيقات الخاصة فهي تستخدم الرمز @ الذي يمثل حرف او مسافة والرمز & الذي يمثل حرف او لا شئ:

```
Print Format("ABCD", "@ @ @ @")                    ' "A B C D"
Print Format("ABCD", "@ &&&")                      ' "A BCD"
Print Format("9661234567", "&&&-&-@@@@@@") ' "966-1-234567"
```

دوال اخرى

من الدوال الحرفية الاخرى دوال استخلاص الحروف اليسرى $Left، الحروف اليمنى $Right و الحروف الوسطى $Mid:

```
Dim sMyString As String

sMyString = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
Print Left$ ( sMyString, 5)          ' ABCDE
Print Right$ ( sMyString, 5)         ' VWXYZ
Print Mid$ ( sMyString, 20, 5)       ' TUVWX
Print Mid$ ( sMyString, 20)          ' TUVWXYZ
```

وعند الحديث عن حروف لغتنا الجميلة، فمن المعروف ان الدالتين Right و Left تعطيان نتائج عكسية مع الحروف العربية. فالاستدعاء $Right للجملة "مرحبا" سبدأ من الالف فالباء فالحاء الخ.. مما يسبب التشويش على المبرمج العربي. الفكرة بكل بساطة لجعل هاتين الدالتين تعملان بشكل صحيح مع الحروف العربية، هي عن طريق تطوير دالتين عربيتين ArRight و ArLeft:

```
Function ArLeft(sString As String, lLength As Long) As String
   ArLeft = Right$(sString, lLength)
End Function

Function ArRight(sString As String, lLength As Long) As String
   ArRight = Left$(sString, lLength)
End Function
```

```
Private Sub Form_Click()
   ' مثال للاستدعاء
   Print ArRight$("تركي العسيري", 4) ' تركي
   Print ArLeft$("تركي العسيري", 7)   ' العسيري
End Sub
```

اعود للدالة $Mid مرة اخرى، فعليك معرفة ان $Mid هي عبارة Statement ايضا، فيمكنك كتابة شيئا مثل:

```
Dim sMyString As String

sMyString = "abcde"
Mid$(sMyString, 2, 3) = "BCD"
Print sMyString          ‘ “aBCDe”
```

وبدلا من معرفة ما اذا كان المتغير الحرفي خاليا باستخدام علامات التنصيص، استخدم الدالة Len التي تعود بعدد حروف القيمة المرسلة فهي اسرع مرتين من الطريقة الاولى:

```
If sMyString = “” Then ...
If Len(sMyString) = 0 Then ... ‘ هذه اسرع
```

اما لحذف المسافات اليمنى استخدم الدالة $RTrim، المسافات اليسرى $LTrim والمسافات اليمنى واليسرى $Trim:

```
sMyString = “  12345  ”
Print RTrim$ (sMyString)      ‘ “  12345”
Print LTrim$ (sMyString)      ‘ “12345  ”
Print Trim$ (sMyString)       ‘ “12345”
```

الدالة Asc تعود بالعدد المقابل للحرف في جدول ASCII، اما AscW فهي للترميز UNICODE، والدوال $Chr و $ChrW تعود بالحرف المقابل للعدد –أي العكس:

```
Print Asc("ت")          ‘ 202
Print AscW("ت")         ‘ 1578
Print Chr$(202)         ‘ ت
```

```
Print ChrW$(1578)    ' ت
```

من الدوال الاخرى دالتي $UCase و $LCase اللتان تقومان بتكبير الحروف الانجليزية وتصغيرها. استخدام هاتان الدالتان مسألة ضرورية خاصة عند مقارنة القيم الحرفية، فلا تنسى ان "Turki" لا تساوي "TURKI":

```
If Text1.Text = "TURKI" Then ...      ' قد لا يكتب المستخدم حروف كبيرة
If UCase$( Text1.Text ) = "TURKI" Then ...
If LCase$( Text1.Text ) = "turki" Then ...
```

الدالة $UCase تقوم بتكبير جميع الحروف والدالة $LCase تقوم بتصغير جميع الحروف كما رأينا في المثال السابق، اما اذا كنت تريد تكبير الحرف الاول من كل كلمة، فارسل الثابت vbProperCase الى الدالة StrConv:

```
sMyString = "I like Visual Basic"
Print StrConv(sMyString, vbProperCase)      ' "I Like Visual Basic"
Print StrConv(sMyString, vbLowerCase)       ' "i like visual basic"
Print StrConv(sMyString, vbUpperCase)       ' "I LIKE VISUAL BASIC"
```

تمكنك الدالة StrConv ايضا من تحويل النصوص من ASCII الى UNICODE بارسال الثابت vbUnicode او من UNICODE الى ASCII بارسال الثابت vbFromUnicode.

> **ملاحظة:** بعض الدوال الحرفية تتوفر منها نسختين، الاولى تعود بقيمة من النوع String والثانية تعود بقيمة من النوع Variant. حاول دائما استخدام النسخة الاولى من الدالة –اذا توفرت- عن طريق اضافة الرمز $ بعد اسم الدالة لترفع عبء تحويل نوع القيمة الذي يقوم به Visual Basic مما يؤدي الى زيادة سرعة التنفيذ.

الدالة Split تمكنك من فصل جميع الكلمات ونسخ كل كلمة الى مصفوفة، افتراضيا الفاصل بين الكلمات هي المسافة كما يمكنك تحديد فاصل معين:

```
Dim X() As String
Dim iCounter As Integer

X = Split(Text1.Text)
```

```
' تحديد فاصل غير المسافة
' X = Split(Text1.Text, "*")

For iCounter = 0 To UBound(X)
   Print X(iCounter)
Next
```

واذا ندمت على تقسيم الكلمات، فالدالة Join تعيد الوضع كما كان سابقا:

```
sFromArrayToString = Join (X)
' sFromArrayToString = Join (X, "*") ' تحديد فاصل غير المسافة
```

اخيرا، دالتي تحويل القيم الى حرفية $Str و CStr، والفرق بينهما هو ان الاولى تضيف مسافة يسار العدد المرسل لها ان كان موجب ام الثانية فلا.

التعامل مع الوقت والتاريخ

لا قتصر دعم Visual Basic على قيم الوقت والتاريخ في امكانية تصريح متغير من النوع Date، بل يحتوي على عشرات الدوال الخاصة بعرض، تعديل، حساب وتنسيق قيم الوقت والتاريخ.

دوال الوقت والتاريخ

القيم من النوع Date هي قيم تمثل وقت معين او تاريخ معين او كلاهما معا سواء كانت في متغير من النوع Date او ثابت بين العلامتين # و #:

```
Dim MyDate As Date

MyDate = #1/20/2001#
Print MyDate
MyDate = #2:30:00 AM#
Print MyDate
MyDate = #1/20/2001 2:30:00 AM#
Print MyDate
```

وقبل ان ابدأ فـي الحـديث عـن دوال الوقـت والتـاريخ، اود ان اعرفـك علـى الخاصـية Calendar التابعة للكائن VBA والتي تمكنك من تغيير نظام التـاريخ التـابع لبرنامجـك الى ميلادي او هجري:

```
Dim MyDate As Date

MyDate = #1/20/2001 2:30:00 AM#
VBA.Calendar = vbCalHijri   ' هجري
Print MyDate
VBA.Calendar = vbCalGreg    ' ميلادي
Print MyDate
```

مع العلم ان القيمة التي تضعها في هذه الخاصية تؤثر على نوع القيمة التـي تعـود بها دوال التاريخ الاخرى ولكنها لا تؤثر في قيم الوقت والتاريخ:

```
Dim MyDate As Date

VBA.Calendar = vbCalHijri
MyDate = #1/16/1421#       ` القيمة هنا بالتاريخ الميلادي وليس الهجري
Print MyDate               ` مخرجات الامر هنا بالهجري
```

والان اعرض لك دوال الوقـت والتـاريخ مبتـدئا بالـدالتين Date و Time اللتـان تعـودان بتاريخ اليوم والوقت الحالي:

```
Print Date
Print Time
```

اما اذا تعاملـت مـع Date و Time كعبـارات Statement، فهـي سـتغير قيمـة الوقـت والتاريخ في النظام:

```
Date = #1/20/2001#
Time = #12:00:00 AM#
```

الدالة Now تعود بقيمة تشـمل تاريخ اليوم والوقت الحالي:

```
Print Now
```

اما الدالة Timer فهي تعود بعدد الثواني من منتصف الليل حتى وقت استدعائها أي هي تعمل كعداد، قد تستفيد منها وتطور اجراء انتظار مؤقت قبل تنفيذ كود معين:

```
Sub Wait ( iSeconds As Integer)
        Dim sStartTime As Single

        sStartTime = Timer
        Do: DoEvents : Loop Until Timer – sStartTime >= iSeconds
End Sub
```

تذكر ان الدالة Timer عبارة عن عداد يبدأ من الساعة 00:00:00 ويتم تصفيره من جديد بعد مرور ثانية من الساعة 23:59:59، فالاجراء Wait السابق قد يؤدي الى حلقة لا نهائية اذا تم تصفير الدالة Timer قبل نهاية الحلقة الموجودة في الاجراء. صحيح ان نسبة حدوث المشكلة السابقة ضئيلة، الا ان تجنب الشوائب امر ضروري، وكما يقولون "ابعد عن الشوائب وغني لها":

```
Sub Wait ( iSeconds As Integer)
        Const NUMOFSEC As Single = 24 * 60 * 60!
        Dim sStartTime As Single
        sStartTime = Timer
        Do : DoEvents
        Loop Until (Timer + NUMOFSEC – sStartTime) Mod NUMOFSEC >= iSeconds
End Sub
```

لن تتمكن من استخدام الثابت بين العلامتين # و # مباشرة اذا كنت ترغب في تعيين قيم لمتغيرات باستخدام التاريخ الهجري، ولكن مع الدالة DateSerial يمكنك عمل ذلك فهي تتأثر بقيمة الخاصية Calendar التابعة للكائن VBA:

```
Dim MyDate As Date

VBA.Calendar = vbCalHijri
MyDate = DateSerial(1422, 10, 27)  ' 27 شوال 1422
Print MyDate
VBA.Calendar = vbCalGreg
Print MyDate  ' يوافق 11 يناير 2002
```

كما تتوفر دالة اخرى للوقت هي الدالة TimeSerial. اما بالنسبة للدالتين DateValue و TimeValue فهما تعودان بقيمة التاريخ او الوقت الموجود في القيمة المرسلة اليهما:

```
Print DateValue(Now + 2)
Print TimeValue(Now)
```

بامكانك استخدام الدالة DatePart التي تستخرج جزء معين من قيمة الوقت او التاريخ، ولكني افضل الدوال Day، Month، Year، Hour، Minute و Second فهي تعود بقيمة اليوم، الشهر، السنة، الساعة، الدقيقة والثانية الموجودة في القيمة المرسلة اليهم:

```
Print Month (Date)
Print Hour (#1:20:00 AM#)
```

العمليات الرياضية على الوقت والتاريخ:

كثيرا ما تحتاج الى اجراء بعض العمليات الرياضية على قيم تاريخية كالجمع بين تاريخين او طرح تاريخين، بالاضافة الى مقارنة التواريخ. بامكانك تطبيق ما ذكرته في فقرة "النوع Date" في الفصل الثاني لان القيمة من هذا النوع –كما ذكرت- تنقسم الى قسمين عدد صحيح وعدد عشري، العدد الصحيح يتعلق بالتاريخ اما العشري فهو خاص بالوقت، جرب استخدام معامل الجمع كما في هذا المثال:

```
' اضافة يومين و 12 ساعة من الان
Print Now + 2 + #12:00#
```

ولتطبيق عمليات رياضية اكثر دقة وسهولة، يفضل استخدام الدالتين DateAdd و DateDiff، الاولى لاضافة تاريخ على تاريخ والثانية لمعرفة الفارق بينهما. دالة DateAdd لها صيغة عامة هي:

```
DateAdd (الفترة, العدد, التاريخ)
```

الفترة هي الوحدة المستخدم والتي قد تكون سنة "yyyy" شهر "m" يوم "d" ....الخ، اما العدد فهو عدد الوحدات من الفترة التي تريد اضافتها، اما التاريخ فهو القيمة الذي تريد اضافة التاريخ عليها:

```
Print DateAdd ("m", 3, Now)
```

اما لمعرفة الفرق بين تاريخين فاستخدم دالة DateDiff وصيغتها العامة شبيهه بالاولى، الا انها تطلب قيمة التاريخ مكان قيمة العدد:

```
Print DateDiff("d", #12/20/2000#, #2/18/2001#) ` 60 يوم
```

تنسيق الوقت والتاريخ

لا اعتقد انني بحاجة الى تعريفك على الدالة Format مرة اخرى، وبالنسبة للتنسيقات القياسية فهي مدعومة لقيم الوقت والتاريخ:

```
Dim sMyDate As Date

sMyDate = Now

Print Format$(sMyDate, "General Date")
Print Format$(sMyDate, "Long Date")
Print Format$(sMyDate, "Long Time")
Print Format$(sMyDate, "HH:MM -> MMMM DD, YYYY")
```

كما اضاف الاصدار VB6 دالة اضافية لتنسيق الوقت والتاريخ هي FormatDateTime والدالة MonthName التي تعود باسم الشهر المقابل للرقم المرسل لها:

```
VBA.Calendar = vbCalHijri
Print MonthName(1)  ` محرم
VBA.Calendar = vbCalGreg
Print MonthName(1)  ` يناير
```

التعامل مع الملفات والمجلدات

لم يضف الاصدار VB6 أي دوال او عبارات جديدة للتعامل مع الملفات والمجلدات، فمعظم ما سأسطره في الفقرات التالية توفر منذ الاصدارات القديمة لـ Visual Basic. وسأبدأ بعرض دوال وعبارات تمكنك من التعامل مع الملفات كتعديل خصائصها، اسمائها، الاستعلام عن احجامها .... الخ، ثم اتطرق الى عبارات خاصة

بالمجلدات Folders وطريقة البحث عن الملفات والمجلدات، ثم اختم الفقـرة بطـرق الوصول الى الملفات وتحريرها.

التعامل مع الملفات

يمكنك الامر Name ... As من اعـادة تسـمية الملـف او نقلـه، والامـر FileCopy مـن نسخ الملف، بينما الامر Kill يحذف الملف:

```
' اعادة تسمية ملف
Name "C:\MyFile.EXT" As "C:\MyFile.DAT"
' نقل ملف
Name "C:\MyFile.EXT" As "D:\MyFile.EXT"
' نسخ ملف
FileCopy "C:\MyFile.EXT" As "D:\MyFile.EXT"
' نسخ وتغيير اسم ملف
FileCopy "C:\MyFile.EXT" As "C:\MyFile2.EXT"
' حذف ملف
Kill "C:\MyFile.EXT"
' حذف مجموعة ملفات
Kill "*.TMP"
```

تمكنك الدالة GetAttr مـن معرفـة خصـائص الملـف File Attributes والدالـة SetAttr لتغيير خصائص الملف –شريطة ان لا يكون مفتوحا:

```
Dim sFile As String

sFile = "C:\File.EXT"
If GetAttr(sFile) And vbHidden Then ... ' مخفي
If GetAttr(sFile) And vbReadOnly Then ... ' للقراءة فقط
If GetAttr(sFile) And vbArchive Then ... ' ارشيف
...
SetAttr sFile, vbHidden                      ' مخفي
SetAttr sFile, vbArchive + vbReadOnly        ' ارشيف وللقراءة فقط
SetAttr sFile, GetAttr(sFile) Xor vbReadOnly ' عكس خاصية للقراءة فقط
```

تعود الدالة FileLen بقيمة تمثل حجم الملـف، والدالـة FileDateTime بوقـت وتـاريخ انشاء الملف:

```
Print FileLen ( sFile )
Print FileDateTime ( sFile )
```

الميزة التي تعجبني في الدالة FileLen هي قابليتها على العودة بحجم الملفات حتى وان كانت مفتوحة، والعيب الذي لا يعجبني في نفس الدالة هو ان القيمة التي تعود بها هي حجم الملف قبل الفتح -أي لن تعود بقيمة صحيحة في حال قيام البرنامج بتغيير حجم الملف.

التعامل مع المجلدات

تعود الدالة $CurDir بقيمة حرفية تمثل الدليل الحالي للقرص الذي ينفذ منه البرنامج او محرك اقراص آخر ترسله الى الدالة:

```
Print CurDir$            ` الدليل الحالي في القرص الحالي
Print CurDir$ ("d")      ` D: الدليل الحالي في القرص
```

وقد ترغب في تغيير الدليل الحالي باستخدام الامرين ChDrive و ChDir، الاول لتغيير القرص والثاني لتغيير الدليل:

```
ChDrive "D:"
ChDir "D:\MsgFolder"
```

ملاحظة:	ان قمت بتغيير الدليل الحالي باستخدام الامر ChDir فقط دون تغيير القرص باستخدام الامر ChDrive، فستقوم بتغيير الدليل الحالي لذلك القرص فقط، اما الدليل الحالي الذي ستعود به الدالة $CurDir لم يتغير.

اذا كنت لا تعرف ما الفائدة من معرفة الدليل الحالي للقرص، فيبدو ان عصر البرمجة تحت النظام MS-DOS لم تشهده اصابعك الرقيقة. بشكل مبسط، الفائدة التي قد تجنيها من تغيير الدليل الحالي هو عدم الحاجة الى تحديد مسار الملفات في ذلك الدليل، فهذا الكود:

```
ChDrive "C:"
ChDir "C:\UnwantedFolder"
Kill "*.*"
```

سـيحذف جميـع الملفـات الموجـودة فـي الـدليل الحـالي للقـرص –وهـو C:\UnwantedFolder، واتمنى من صـميم قلبـي الحنـون ان لا تجعـل دليـل النظـام Windows هو الدليل الحالي وتطبق الكود السابق.

البحث عن الملفات والمجلدات

تمكنـك الدالـة Dir مـن البحـث عـن الملفـات والمجلـدات. طريقـة اسـتخدامها يـتم بخطوتين: الاولى تحديد الملف/الملفات/المجلد وخصائصها، والثانية باستدعاء الدالة دون ارسـال أي قيمة لها، الكود التالي يبحث عن جميع الملفات التنفيذية الموجـودة في المجلد C:\WinNT :

```
Dim sFileName As String

' الخطوة الاولى
sFileName = Dir$ ("C:\Winnt\*.EXE")
' الخطوة الثانية
Do While Len (sFileName)
        List1.AddItem sFileName
        sFileName = Dir$
Loop
```

تحرير الملفات

بالاضافة اوامر ودوال الاسـتعلام عن الملفات والمجلدات السـابقة، تـوفر لـك مكتبـات VB و VBA اوامر ودوال اخرى تمكنك من تحرير الملفـات لحفـظ بيانـات برامجـك فيهـا بالتنسـيق والهيئة التي تريدها. قبل اجراء أي عمليات تحرير على الملـف، لابـد مـن فتحه باستخدام العبارة Open التي ضيغتها:

```
Open اسم الملف For نوع الوصول Lock الاقفال As #رقم الملف
```

بالنسـبة لرقم الملف، فهو رقم يمثل الملف بحيث يمكنك الوصول اليه من كافة انحاء البرنامج، ولا يمكن لهذا الرقم ان يشـمل اكثر مـن ملـف واحـد، لـذلك حتـى تتفـادى اخطاء التعارض، يفضل اسـتخدام الدالة FreeFile التي تعود بـرقم غيـر محجـوز لفـتح الملف. وبالنسـبة للاقفال، فهي تمكنك من تحديد خاصية الاقفال على الملف بحيث يمكنك منع كافة البرامج الاخـرى مـن القـراءة مـن الملـف Lock Read، الكتابـة الـى

الملف Lock Write او القراءة والكتابة من والـى الملـف Lock Read Write. اما نـوع الوصول، فهي الطريقة التي تود ان تتعامل مع الملف بها وهي ثلاثة انواع:

الوصول المتتالي Sequential Access:

الاسـلوب المتبع مع الوصـول المتتـالي يعـرف بـالقراءة والكتابـة سـطر سـطر. ولفـتح الملـف، اسـتخدم الكلمـة المحجـوزة Input للقـراءة مـن الملـف، الكلمـة المحجـوزة OutPut للكتابة الى الملف والكلمة المحجوزة Append للاضافة الى الملف:

```
Open "MyFile.TXT" For Input As #1
Open "MyFile2.TXT" For OutPut As #2
Open "MyFile3.TXT" For Append As #3
```

بامكانك قراءة سطور من الملفات -المفتوحـة بالكلمـة المحجـوزة Input- باسـتخدام العبارة Line Input حتى نهاية الملف والذي تختبره عن طريق الدالة EOF:

```
Dim sLine As String

Open "MyFile.TXT" For Input As #1
Do While Not EOF(1)
	Line Input #1, sLine
	Text1.Text = Text1.Text & vbNewLine & sLine
Loop
```

الكود السـابق لا استخدمه كثيرا فأنا افضل قراءة الملف كاملا بدالـة واحـدة تسـمى Input$، واستخدم في ذلك الدالة LOF التي تعود بالحجم الكلي للملف:

```
Dim sFileData As String

Open "C:\MyFile.TXT" For Input As #1
sFileData = Input$ (LOF(1), 1)
Text1.Text = sFileData
```

وبامكانـك كتابـة سـطور الـى الملفـات -المفتوحـة بالكلمـة المحجـوزة OutPut و Append- باستخدام العبارة # Print:

```
Open "C:\MyFile.TXT" For Append As #1
```

```
Print #1, Text1.Text
```

ولا تنسى اغلاق الملف باستخدام العبارة Close التي ستغلق كافة الملفات المفتوحة ان لم ترسل لها رقم ملف معين:

```
' اغلاق الملف رقم 1
Close #1
' اغلاق كآفة الملفات
Close
```

الوصول الثنائي Binary Access:

الاسلوب المتبع مع الوصول الثنائي يعرف بالقراءة والكتابة بايت بايت. ولفتح الملف، استخدم الكلمة المحجوزة Binary للقراءة والكتابة من والى الملف:

```
Open "C:\MyFile.DAT" For Binary As #1
Open "D:\YouFile.DAT" For Binary As #2
```

عملية القراءة والكتابة من الملف متشابهتان من ناحية الصيغة الى حد كبير. كل ما هو مطلوب منك هو معرفة الموقع في الملف وحجم العملية. عندما تقوم بفتح الملف لاول مرة، فان موقع مؤشر القراءة والكتابة من الملف هو 1، وهو اول بايت موجود في خارطة الملف. لمعرفة موقع اخر بايت استخدم الدالة LOF والتي تعود بحجم الملف والذي بديهيا يرمز الى موقع البايت الاخير:

```
Print LOF (1)
Print LOF (2)
```

الذي كنت اقصده من "حجم العملية" هو حجم البايتات التي تريد قراءتها من الملف او كتابتها الى الملف. كل هذا يتم باستخدام الامر Get للقراءة او Put للكتابة. راقب هذا الكود:

```
Dim X As Long

Get #1, 1, X
Print X
```

```
Get #1, , X ` 5 موقع المؤشر هو
Print X
```

من المهم التنويه هنا بان عملية القراءة من الملف تؤدي الى زيادة الموقع الحالي للمؤشر بمقدار حجم العملية. ففي السطر الثاني لم احدد موقع المؤشر، لانه سيزيد بشكل تلقائي 4 بايتات وذلك بسبب ان حجم العملية السابق = 4 بايت لقراءة قيمة من نوع Long. هذا الكود يقوم بقراءة جميع الارقام من ملف وكتابتها في ملف اخر:

```
Dim lCounter As Long
Dim X As Long

For lCounter = 1 To LOF(1)
   Get #1, , X
   Put #2, , X
Next
```

يمكنك تغيير موقع مؤشر الملف عن طريق العبارة Seek:

```
Seek #1, 1              ` الى بداية الملف
Seek #2, LOF(2)         ` الى نهاية الملف
```

بدون شك تحتاج الى التعامل مع القيم الحرفية Strings والتي تتم بنفس الطريقة لكنك بحاجة الى اعطاء Visual Basic معلومات عن حجم القيمة الحرفية. يمكنك عمل ذلك؟ باستخدام النوع الحرفي الثابت الحجم Fixed Length String. كما في المثال التالي والذي سيقرأ 100 بايت من الملف:

```
Dim Y As String * 100

Get #1, , Y
```

واذا كنت لا تفضل استخدام هذا النوع من المتغيرات، فيمكنك عمل أي شئ تخبر فيه Visual Basic ان حجم المتغير الحرفي هو 100 وذلك عن طريق اسناد أي قيمة مؤقتة:

```
Dim Y As String
```

```
Y = String (100, "*")

Get #1, , Y
```

وعملية كتابة القيم الحرفية الى الملف يمكن لها ان تتم بشكل مباشر مثل:

```
Put #1, , "ملف ثنائي"
```

تذكر دائما ان النوع String هو ترميز احادي في الذاكرة مما يؤدي الى بطئ عملية التحويل خاصة في حالة كون القيم الحرفية كبير جدا. ولزيادة السرعة اكثر من 50%، استخدم عملية المصفوفات للنوع لهذا النوع من القيم -النوع Byte بدلا من String:

```
Dim MyArray(1000) As Byte

Open "MyFile.DAT" For Binary As #1

' كتابة محتويات المصفوفة الى الملف
Put #1, 1, MyArray

' او قراءة محتويات الملف الى المصفوفة
Get #1, 1, MyArray
```

الوصول العشوائي Random Access:

الاسلوب المتبع مع الوصول العشوائي يعرف بالقراءة والكتابة سجل سجل. ولفتح الملف، استخدم الكلمة المحجوزة Random للقراءة والكتابة من والى الملف مع ارسال حجم السجل:

```
Open "C:\MyData.DAT" For Random As #1 Len = 200
```

استخدم العبارة Put للكتابة الى الملف والعبارة Get للقراءة من الملف كما كنت تفعل مع الملفات الثنائية، ولكن عليك معرفة ان حجم العملية وخطوات انتقال المؤشر تتأثر بالحجم المصرح عند فتح الملف باستخدام الكلمة Len.

يفيدك هذا النوع من الملفات لمحاكاة قواعد البيانات بطريقة مبسطة، مـثلا يمكنـك تعريف تركيب UDT والكتابة الى الملف:

```
Private Type typRCD
   sName As String * 20
   iAge As Integer
End Type

Dim Record As typRCD
Open "C:\MyData.DAT" For Random As #1 Len = Len(Record)
Record.sName = "تركي"
Record.iAge = 99
Put #1, 1, Record
Record.sName = "عبدالله"
Record.iAge = 20
Put #1, , Record
```

وقراءة السجلات تتم بهذه الطريقة:

```
Dim Record As typRCD
Get #1, 1, Record
Do While Not EOF(1)
Print Record.sName
Print Record.iAge
Get #1, , Record
Loop
```

كائنات اخرى

الى جانب الدوال والاجراءات السابقة، توفر لك مكتبات VB و VBA مجموعة لا غنـى عنها من الكائنات المستخدمة في برامجك الجدية.

كائن البرنامج App

كائن البرنامج App يمثل البرنامج الحالي الذي يتم تنفيذه. يحتوي على مجموعة من الخصائص والطرق التي سأتطرق الى بعضها هنا، اما البقية فهي متقدمة بعض الشئ وافضل تأجيلها الى الفصول اللاحقة.
الخاصية EXEName تعود باسم ملف البرنامج التنفيذي EXE، والخاصية Path تعود بالمسار الكامل الذي نفذ البرنامج منه:

```
Open App.Path & "\" & App.EXEName & ".EXE" For Binary As #1
```

من الضروري التحقق من الرمز "\" قبل استخدام الخاصية Path، ففي الكود السابق اضفنا هذا الرمز بعد الخاصية Path وذلك لان مسار البرنامج لن يضاف اليه هذا الرمز، ولكن تظهر المشكلة في الكود السابق اذا تمت عملية تنفيذ البرنامج من الدليل الجذري Boot Directory للقرص، انظر الى الرمز "\" في قيمة الخاصية Path اذا نفذ البرنامج من دليل جذري او فرعي:

```
App.Path = "C:\"                ' الرمز مضاف
App.Path = "C:\MyProgram"  ' الرمز غير مضاف
```

وحتى تتجنب المشكلة السابقة، طور هذه الدالة وحاول الاعتماد عليه عوضا عن الخاصية Path مجردة:

```
Function PathEx() As String
    If Right(App.Path, 1) = "\" Then
        PathEx = App.Path
    Else
        PathEx = App.Path & "\"
    End If
End Function

Open PathEx & App.EXEName & ".EXE" For Binary As #1
```

الخاصية PrevInstance تمكنك من معرفة ما اذا كانت نسخة اخرى من البرنامج التنفيذي قيد العمل او لا، قد تستطيع منع المستخدم من تشغيل اكثر من نسخة للبرنامج في نفس الوقت بهذا الكود:

```
If App.PrevInstance Then
        MsgBox "لا يمكنك تشغيل نسخة اخرى من البرنامج"
        End
End If
```

مع ذلك، لا تثق في الكود السابق كثيرا، فالمستخدم بامكانه تشغيل اكثر من نسخة من نفس البرنامج اذا قام بنسخ ملف البرنامج الى مجلد اخر او حتى الى نفس المجلد باسم آخر.

من الخصائص التي يمكنك تعديل قيمها الخاصية TaskVisible التي تمكنك من اخفاء او اظهار اسم او رمز البرنامج في قائمة البرامج Task List- وهي النافذة التي تمكنك من عرض جميع البرامج العاملة عن طريق الضغط على المفاتيح [Ctrl+Alt+Del] او [Ctrl+Shift+ESC]، فلاخفاء اسم البرنامج في أي وقت:

```
App.TaskVisible = False
```

وكذلك الخاصية Title التي تمكنك من تحديد النص الظاهر في قائمة البرامج Task List، يكون النص الافتراضي هو النص الموجود عند خانة اسم المشروع Project Name في صندوق حوار خصائص المشروع Project Properties قبل عملية الترجمة.

من الخصائص الاخرى التي تجدها في صندوق الحوار السابق والكائن App هي خصائص رقم اصدار البرنامج Major، Minor .... الخ، وخصائص حقوق الملكية LegalCopyRight، Trademarks .... الخ وهي للقراءة فقط وقت التنفيذ.

كائن الحافظة Clipboard

تتميز معظم تطبيقات Windows بامكانية الاتصال وتبادل البيانات فيما بينها، صحيح ان تبادل البيانات عن طريق الحافظة Clipboard محدود الامكانيات، الا انه اسلوب مازال متبع في معظم تطبيقات Windows. يمكنك Visual Basic من نسخ ولصق البيانات من والى الحافظة عن طريق الكائن Clipboard.

نبدأ أولاً بنسخ النص الى الحافظة باستخدام الطريقة SetText:

```
Clipboard.Clear
Clipboard.SetText Text1.Text, vbCFText
```

ملاحظة: ينصح دائما باستخدام الطريقة Clear لمسح محتويات الحافظة قبل نسخ البيانات لها، وذلك لانه في حالات معينة لن تتمكن من نسخ بيانات جديدة الى الحافظة مالم يتم مسح محتوياتها.

ولنسخ النصوص مع تنسيقها على هيئة RTF فاستخدم الثابت vbCFRTF:

```
Clipboard.Clear
Clipboard.SetText RichTextBox1.Text, vbCFRTF
```

اما لنسخ الصور، فالطريقة SetData هي المستخدمة:

```
Clipboard.Clear
Clipboard.SetData Picture1.Picture
```

ولاجراء عملية لصق النصوص، فالطريقة GetText هي المستخدمة، ولكن عليك اختبار نوع وهيئة البيانات الموجودة في الحافظة باستخدام الطريقة GetFormat قبل القيام بعملية اللصق:

```
If Clipboard.GetFormat (vbCFText) Then
        Text1.SelText = Clipboard.GetText (vbCFText)
ElseIf Clipboard.GetFormat (vbCFRTF) Then
        RichTextBox1.SelRTF = Clipboard.GetText (vbCFRTF)
End If
```

ولصق الصور استخدم الطريقة GetData التي تشترط هيئة الصورة:

```
If Clipboard.GetFormat(vbCFBitmap) Then
        Set Picture1.Picture = Clipboard.GetData(vbCFBitmap)
End if
```

رغم ان أداة الصورة PictureBox يمكنها عرض انواع وهيئات مختلفة من الصور، الا ان الكود السابق لن يعمل الا اذا كانت هيئة الصورة من النوع vbCFBitmap، لذلك يفضل تمكين جميع الهيئات الاخرى:

```
Dim picFormat As Variant
```

```
For Each picFormat In Array(vbCFBitmap, vbCFMetafile, vbCFDIB, vbCFPalette)
        If Clipboard.GetFormat(picFormat) Then
                Set Picture1.Picture = Clipboard.GetData(picFormat)
                Exit For
        End If
Next
```

كائن الشاشة Screen

كائن الشاشة يمثل جميع شاشات ونوافذ برنامجك ويحتوي على خصائص تتعلق بالمظهر العام لسطح مكتب نظام التشغيل، كالخاصية FontCount التي تعود بمجموع الخطوط المتوفرة في نظام التشغيل والتي تستخدمها مع الخاصية Font التي تعود باسماء الخطوط:

```
Dim iCounter As Integer
For iCounter = 0 To Screen.FontCount - 1
        List1.AddItem Screen.Fonts(iCounter)
Next
```

الخاصية ActiveForm تمثل نافذة النموذج النشطة في البرنامج، وهي مرجع الى كائن نافذة النموذج، اما الخاصية ActiveControl فهي تمثل الاداة التي عليها التركيز:

```
Screen.ActiveForm.Caption = "النافذة النشطة"
```

اخيرا، الخاصيتان Height و Width تعودان بارتفاع وعرض الكثافة النقطية Resolution للشاشة:

```
Print "Width = " & ScaleX(Screen.Width, vbTwips, vbPixels)
Print "Height = " & ScaleY(Screen.Height, vbTwips, vbPixels)
```

كائن الطابعة Printer

الطباعة من المزايا الضرورية التي لابد من توفيرها في برامجك الجدية. بعيدا عن اجراءات API المعقدة، يوفر لك Visual Basic كائن الطابعة Printer Object والذي من اسمه يعرف غرضه.

قبل التوغل في اعضاء كائن الطابعة اود ان اتطرق الى مجموعة الطابعات Printers Collection. هذه المجموعة تمثل جميع الطابعات الموجودة في الجهاز الحالي. لاتستطيع تعديل مزايا هذه الطابعات بشكل مباشر. فلابد في البداية من تحديد الطابعة وتعيينها للاستخدام ومن ثم تستطيع تعديل الخصائص.
من الخصائص الموجودة في مجموعة الطابعات Printers خاصية اسم الطابعة DeviceName او اسم المشغل DeriverName، رقم المنفذ Port المركب عليه الطابعة ....الخ:

```
Dim X As Integer

' استخدام المجموعة
' Printers Collection
For X = 0 To Printers.Count -1
  Print Printers(X).DeviceName
Next
```

بكل تأكيد تود من المستخدم تحديد الطابعة التي يريد استخدامها. كل ما عليك هو توفير كود مناسب لتعيين الطابعة الموجودة في المجموعة Printers الى الكائن Printer. هذا مثال لعمل ذلك باستخدام الاداة ListBox:

```
Private Sub Form_Load()
  Dim X As Integer

' استخدام المجموعة
' Printers Collection
  For X = 0 To Printers.Count -1
    List1.AddItem Printers(X).DeviceName
  Next
End Sub

Private Sub List1_Click()
' تحديد الطابعة من المجموعة
  Set Printer = Printers(List1.ListIndex)
End Sub
```

والان كل ما عليك هو استخدام الكائن Printer لتعديل خصائص الطابعة الحالية او البدء في عملية الطباعة. من هذه الخصائص، خاصية ColorMode التي تعرف عن طريقها ما اذا كانت الطابعة داعمة للالوان ام لا. والخاصية PrinterQuality والتي تعود بنوع الكثافة النقطية وجودة الطباعة. الخصائص الاخرى تجدها بشكل مفصل في مكتبة MSDN.

اما عملية الطباعة الفعلية فتتم باستخدام طرق كائن الطابعة وهي نفس الطرق الموجودة في كائن النموذج Print، Line، Cirlce .... الخ واستخدامها يتم بنفس الطريقة التي استخدمناها مع نافذة النموذج في الفصل الثاني "النماذج والادوات". بعد ان تنتهي من ارسال البيانات الى الطباعة، استخدم الطريقة EndDoc لبدء عملية الطباعة الفعلية:

```
Printer.RightToLeft =True
Printer.FontSize = 40
Printer.Print "تركي العسيري"
' ابدأ عملية الطباعة
Print.EndDoc
```

الطريقة KillDoc تقوم بانهاء عملية الطباعة، والطريقة NewPage واضح من اسمها انها تخرج صفحة جديدة.

اكتشاف الاخطاء

مما لا شك فيه، ان من اهم اسباب انتشار الشعيرات البيضاء في رؤوس المبرمجين هي الاخطاء البرمجية. فكم من مبرمج كسر شاشة جهازه بسبب كثرة الاخطاء غير المتداركة في برنامجه، وكم من مبرمج توقف عن إكمال مشاريعه بسبب عدم معرفة مصدر الخطأ. كتابة برنامج دون اخطاء شيء يتحقق في الخيال فقط! لكن كلما زادت احتياطاتك لتفادي الاخطاء قلت نسبة ظهورها.

فكرة عامة

تصنف الاخطاء في أي لغة برمجة الى صنفين على اساس وقت حدوثها اما في وقت التصميم او وقت التنفيذ. هذه الاخطاء تسبب انهيار برنامجك وانهاء تنفيذه. بالاضافة الى ذلك، يوجد نوع من الاخطاء التي لا تظهر لك بشكل مباشر تعرف بالشوائب Bugs. لنلقي نظرة على هذه الاصناف:

اخطاء وقت التصميم Design Time Errors:

وتعرف ايضا بالاخطاء النحوية Syntax Errors وهي اسهل انواع الاخطاء اكتشافا واصلاحا. وقت حدوث هذه الاخطاء يكون في مرحلة التصميم او الترجمة للبرنامج. سببها الرئيسي في طريقة كتابة العبارات البرمجية الخاطئة. فمثلا قد تكتب اسم دالة ليست موجودة، او تنشئ حلقة For بدون اقفالها باستخدام Next.

توفر لك بيئة التطوير المتكاملة لـ Visual Basic تقنية في قمة الروعة هدفها قنص هذه الاخطاء تلقائيا بمجرد الوقوع فيها وذلك بعد الضغط على المفتاح [ENTER]. مثلا، قم بكتابة 4 == X واضغط مفتاح [ENTER] ستلاحظ ظهور رسالة توضح لك الخطأ وقد قلب لون السطر بالاحمر. تعرف هذه التقنية بالتدقيق النحوي التلقائي Auto Syntax Check والتي تستطيع الغائها عن طريق الاختيار Auto Syntax Check الموجود في خانة التبويب Editor في نافذة الخيارات Options. لا اعتقد انك ستلغيها يوما من الايام اليس كذلك؟!

اخطاء وقت التنفيذ Run Time Errors:

وقت ظهور هذه الاخطاء مختلف. فلن تظهر الرسالة المزعجة السابقة وقت كتابة الكود وانما في وقت التنفيذ. عندما يصل المفسر عند سطر صحيح نحويا لكنه خاطئ منطقيا ستظهر رسالة خطأ بعنوان Run Time Error ويظهر تحديد لمكان السطر الذي وقع فيه الخطأ. مثلا اكتب هذا الكود:

```
Dim X As Byte
X = 256
```

من الواضح ان الصيغة النحوية لهذا الكود صحيحة لكن من الناحية المنطقية خطأ. جرب تنفيذ البرنامج وستلاحظ ظهور رسالة خطأ OverFlow وذلك لان القيمة القصوى التي يمكن ان يحملها أي متغير من نوع Byte هي 255. طبعا اخطاء وقت التنفيذ كثيرة جدا جدا، فانت عندما تصمم البرنامج تتوقع ان كل الاحتمالات الخارجية كما هي في حالة تصميم البرنامج. مثلا لو وجد في احد سطور برنامج امر يقوم بمسح ملف معين وكتبت هذا الكود:

```
Kill "FileName.EXT "
```

قد عمل معك بالشكل المطلوب، لكن افترض ان الملف لم يكن موجود؟ فان رسالة الخطأ سيكون لها نصيب من عمر تنفيذ البرنامج. فلو كنت ذكيا ستتأكد من وجود الملف باستخدام دالة Dir ومن ثم حذفه:

```
If Dir$("FileName.EXT") Then Kill "FileName.EXT "
```

يبدو ان ذكائك خارق جدا يا قارئ هذه السطور لكن مهلا كاتب هذه السطور لديه شئ اخر ليخبرك به. ماذا لو كان الملف موجود لكن خاصية ReadOnly مدعومة به أي انه غير قابل للحذف؟؟ ارأيت رسالة الخطأ ستظهر من جديد. اذن ستستخدم ذكائك الخارق وتتأكد من وجود الملف ومن ثم من عدم وجود خاصية ReadOnly:

```
If Dir$("FileName.EXT") Then
   If Not (GetAttr("FileName.EXT") And vbReadOnly) Then
      Kill "FileName.EXT"
   End If
End If
```

حسنا يا قارئي الفاضل، لك مني فائق التقدير والاحترام على محاولتك الرائعة لتجنب الخطأ لكن مع ذلك فهناك احتمال اخر لوقوع الخطأ! افترض ان الملف FileName.EXT يتم استخدامه من قبل برنامج اخر Process وكانت عليه خاصية الاقفال -أي مسموح للبرنامج الذي يستخدمه فقط- فانك لن تستطيع فتح الملف وستظهر رسالة الخطأ التي اخبرتك بها واكون قد غلبتك في هذا التحدي.
القضية ليست من يغلب من، فكلنا مبرمجين ننسى كثيرا. لكن لابد لأي مبرمج من وضع جميع وكافة الاحتمالات الممكنة لتفادي وتجنب الاخطاء كما سنرى لاحقا.

الشوائب Bugs:
قد يكون الكود سليم من ناحية نحوية ولا توجد به أي اخطاء في وقت التنفيذ لكن به شوائب. لا يوجد برنامج الا وبه شوائب. الشوائب هي اخطاء في سلوك تنفيذ البرنامج لكنها لا تسبب في ايقافه، وهي صعبة الايجاد والاكتشاف. لذلك، تجد غالبا في البرامج التجارية الكبيرة صدور نسخ تجريبية Beta توزع على اشخاص وشركات معينة الهدف منها تجربة البرنامج والتحقق من واكتشاف الشوائب الموجودة فيه. من اكبر الاخطاء الذي يقع فيها المبرمج هي محاولة اكتشاف الشوائب بنفسه، لأنك لن تستطيع اكتشاف الشوائب الا عن طريق غيرك، ففي حالة تجربة برامجك الشخصية فانك اكثر من يعرف طريقة التعامل معها، لكن في حالة وجود نسخة من البرنامج عند شخص اخر فالوضع يختلف، وتبدأ الشوائب بالظهور لديه.
Visual Basic به شوائب. هناك الكثير من الشوائب التي تكتشف شهريا وتصدر شركة Microsoft تقارير عنها تجدها بشكل دوري في مكتبة MSDN، بعضها تم

اصلاحه والبعض الاخر لا. المقصد من هذا الكلام، انه مهما كان مستواك في البرمجة لابد من وجود شوائب في برنامجك.
يوجد الكثير من الكتب التي تناقش مسألة الشوائب البرمجية وكيفية تفاديها -اقصد الاقلال منها- الا انها مواضيع خارج نطاق الكتاب.

الكائن Err

عودا الى موضوع اخطاء وقت التشغيل وبالتحديد في مثال الملف FileName.EXT، بدلا من كتابة عشرات الاسطر للتأكد من قابلية حذف الملف، استخدم كائن الخطأ Err. قبل تطبيق هذا الكائن، عليك معرفة أن كل خطأ من اخطاء وقت التشغيل له رقم خاص يميزة عن غيرة من الاخطاء به وكذلك وصف نصي مختصر للخطأ، وعند حدوث الخطأ يتم وضع هذه البيانات -الخاصة بالخطأ- في الكائن Err. عند رغبتك في الاستمرار في عملية تنفيذ البرنامج حتى عند وقوع الخطأ لابد من كتابة التعليمة On Error Resume Next عند بداية كل اجراء تتوقع حدوث خطأ فيه حتى يستمر في تنفيذ سطور البرنامج راقب هذا المثال:

```
On Error Resume Next

Kill "FileName.EXT"
If Err Then
   MsgBox Err.Description
   Err.Clear
End If
```

هنا سنقوم بمحاولة حذف الملف، ان لم يستطع البرنامج فعل ذلك فان الكائن Err سيحتوي على خصائص تتعلق بذلك الخطأ وسنظهر رسالة توضح وصف الخطأ. من المهم التأكد من تنظيف الكائن Err عن طريق استدعاء الطريقة Clear حتى نخبر البرنامج اننا انتهينا من قنص الخطأ وانه لا يوجد خطأ اخر.
اما اذا كانت اكواد الاجراء طويلة ولا تود ان تكتب الجملة الشرطية If Err Then مرات متعددة، فيفضل استخدام On Error Goto X والتي تؤدي الى الانتقال الى سطر معين في حال حدوث أي خطأ في تنفيذ اكواد الاجراء:

```
Sub LongSub ()
        On Error Goto X:

        …
        …
        …

X:
        MsgBox Err.Description
        Err.Clear
End Sub
```

الفصل الخامس
البرمجة كائنية التوجه OOP

عرّفت الاصدارات VB1، VB2 و VB3 ان Visual Basic هـي لغـة برمجـة مبنيـة علـى الكائنات Object Based Programming Language –تختصر OBP، اما نقطة التحـول كانت منذ انطلاق الاصدار VB4 والذي مكننا من اعتبار Visual Basic علـى انهـا لغـة برمجة كائنيـة التوجـه Object Oriented Programming Language –تختصـر OOP- بعد اضافة ميزة تعريف الفئات Classes على هذه اللغة، الا ان البعض يعتـرض علـى وصف Visual Basic بانها OOP لعدم دعمها لـبعض الميـزات الاساسـية للغـات OOP كالوراثة Inheritance، إعادة التعريف OverLoading .... الخ، مـن ناحيـة اخـرى فهـو يدعم المبدأ الاساسـي للغات OOP وهو التغليف Encapsulation عن طريـق تعريـف الفئات Classes والواجهات Interfaces.
خلاصة القول، Visual Basic لا يعتبر لغة كائنية التوجـه OOP حقيقيـة كلغـات C++، SmallTalk او JAVA، ولكنه يمكنك من محاكاة لغـات OOP وتطبيـق معظـم مبادئهـا. ومن منطلق عنوان هذا الكتاب "Visual Basic للجميع - نحو برمجة كائنيـة التوجـه"، فهذا الفصل هو مدخلك الرئيس الى البرمجة كائنة التوجه OOP وتطبيقها بـ Visual Basic، وسـتكون جميع الفصول اللاحقة من هذا الكتاب مبنية على هذا الفصل.
نظراً لأن جميع الفصول اللاحقة من هذا الكتاب سـتكون مبنية على مـا تعلمتـه مـن هذا الفصل، فاني ارجو منك أن تتقبل مني هاتين النصيحتين:

- لا تحاول تجـاوز إي فقـرة. اذا شـعرت أنـك غيـر مسـتوعب للفكـرة، حـاول قـراءة الفقرة جديد مع تطبيق الأمثلة المدرجة.
- حاول ابتكار امثلة جديدة من وحي افكارك، وقـم بتطبيقهـا، لتتمـرس علـى هـذا الأسلوب من البرمجة.

مقدمة الى OOP

اذا كنت على دراية كافية بمصطلح البرمجة كائنية التوجـه OOP فتسـتطيع الانتقـال الى فقرة بناء اول فئة مبسطة، اما ان كنـت جديـدا علـى OOP فيمكننـي ان اعـرف لك البرمجة كائنية التوجه على انها برمجة موجهه نحو كائنات او اهداف، فكل شـئ

في برنامجك عبارة عن كائن Object او شئ Thing له بيانات وافعال خاصة به أي اشبه بالعالم الحقيقي الذي تراه يوميا، فالانسان كائن له صفات معينة (خصائص Properties) كالاسم، العمر، اللون، الطول، الوزن، .... الخ، وله افعال يقوم بها (طرق Methods) كالمشي، الكتابة، الضحك، البكاء، النوم، .... الخ، كما ان الانسان تحدث عليه وقائع (احداث Events) تؤثر فيه وينتج عنها ردود فعل كاستقبال رسالة مفرحة او محزنة، التعرض لجلطة في المخ، وصول لكمة خطافية في الفك الايمن، صفعة قوية في الخد الايسر، .... الخ.
كذلك الحال مع كائنات Visual Basic، فهي تحتوي على خصائص تحوي بيانات خاصة بها مثل: Left، Height، BackColor .... الخ، وطرق لتفعل افعال خاصة بها مثل: Move، Refresh، ZOrder .... الخ، واحداث تقع عليها كـ Click، MouseMove، KeyPress، .... الخ تنتج عنها ردود فعل خاصة.

لماذا OOP؟

بصراحة الفوائد التي تجنيها من OOP كثيرة جدا ولكني ساختصر ثلاثة منها:

- عندما تكبر حجم البرامج تزداد عملية ادارتها تعقيدا، لكن مع OOP فالبرنامج يتكون من مجموعة كائنات بحيث انه لو حدثت مشكلة في احدها فكل ماهو مطلوب هو تعديل ذلك الكائن دون ان تتأثر الكائنات الاخرى، وحتى لو اردت تطوير احد الكائنات فلست مضطرا الى تنقيح آلاف الاسطر من البرنامج، وكل ما يتوجب عليك القيام به هو الانتقال الى كود الفئة وتطويره فقط.

- تصميم البرامج والتخطيط لبنيتها اصبحت اسهل من البرمجة الاجرائية واقرب الى العالم الحقيقي، فعندما تخطط لبرنامج جديد فنظرتك ستكون بعيدة عن الاكواد وقريبة الى التصميم بحيث تنجز مهامك بسرعة اكبر وسهولة اكثر. فعندما تصمم فئة جديدة، فلن يشغلك أي كود او متغير خارج هذه الفئة قد يؤثر على سلوك تنفيذ الاكواد، وسيكون كل تركيزك على هذه الفئة وكأنها الجزء الوحيد الموجود في البرنامج، مما يقلل نسبة الشوائب Bugs وعدم تأثر متغيرات وبيانات برنامجك.

- OOP ستجعل حياتك اسهل، فلو تخيلت Visual Basic دون كائنات واردت تغيير اسم النافذة، فقد تكتب شيئا مثل:

```
Dim hWND As Long
hWND = FindWindow ("Form1")
ChangeCaption ( hWND, "Main Menu")
```

لكن مع OOP فانك تتحدث عن كائن اسمه Form1 وتقوم بتغيير خاصيته Caption الى الاسم الذي تريده بسهولة شديدة.

سمات OOP

من الضروري ان اوضح الفرق بين الفئة Class والكائن Object، فالفئة –بشكل مبسط- هي مجرد وصف لخصائص، طرق واحداث الكائن، بينما الكائن هو وحدة تحتوي على بيانات واكواد معرفة في الفئة. اعود للمثال السابق، فالانسان هو فئة خلقها الله عز وجل واصفة لخصائص، طرق واحداث كائنات مشتقة منها، فأنا –واعوذ بالله من كلمة انا- كائن لدي خصائص من فئة الانسان كالاسم تركي، العمر 99 .... الخ، وانت ايضا كائن لديك خصائص من نفس الفئة "الانسان" كاسمك س، عمرك ص .... الخ. كذلك الحال مع Visual Basic، فادوات النص Text1 و Text2 هي كائنات من الفئة TextBox، وادوات العنوان Label1، Label2 و Labe3 هي كائنات من الفئة Label.

بودي ان اعرض عليك باختصار السمات الثلاث لـ OOP:

التغليف:

يقصد بالتغليف Encapsulation في لغات OOP بوضع جميع الاشياء معا Putting everything together، بحيث تحقق استقلالية الكائن المطلقة ببياناته الخاصة به وحتى اكواده، من المزايا التي يقدمها لك التغليف هو امكانية تطوير البنية التحتية للكائن بدون ان يتأثر تركيب برنامجك ودون الحاجة الى تعديل سطر واحد من اكواد البرنامج، مثلا لو قمت بتصميم فئة للبحث عن الملفات واعتمدت عليه بدرجة كبيرة في برنامجك، وبعد فترة من الاختبارات والتجارب القوية لاحظت بطء في عملية التنفيذ، فكل ما ستفعله هو تعديل البنية التحتية للفئة الخاصة بالبحث وتطوير خوارزميات اكوادها دون تغيير سطر واحد من سطور البرنامج الاخرى والتي تستعمل هذه الفئة بالتحديد.

كلما زادت استقلالية الفئة، كلما زادت كفاءة اعادة استخدامها في برنامج آخر وتطبيق اسلوب اعادة استخدام الاكواد Code Reusability. مبدأ اعادة استخدام الاكواد من احد المبادئ الضرورية التي يتوجب عليك محاولة والتعود على تطبيقها دائما في برامجك ومشاريعك اليومية، بحيث تتمكن من الاستفادة من الفئة التي صممتها في اكثر من مشروع واكثر من برنامج. وحتى تنشئ فئة قابلة لاعادة الاستخدام، حاول دائما وقبل ان تبدأ بكتابة سطر واحد من الفئة باخذ احتياطاتك

للمستقبل واسأل نفسك اسئلة شبيهه بـ: كيف يمكنني الاستفادة من هذه الفئة في برنامج آخر؟ كيف اسمي واحدد الخصائص، الطرق والاحداث بحيث تكون قابلة للعمل مع اكثر من برنامج وقابلة للتطوير ايضا ؟ كيف اجعل هذه الفئة مستقلة قدر المستطاع عن أي اكواد او كائنات اخرى في البرنامج بحيث يمكنني استخدامها في برنامج آخر؟ ....الخ من الاسئلة والاعتبارات التي لابد من وضعها في الاعتبار قبل بناء الفئة وعند كتابة كل اجراء من اجراءاتها.

تعدد الواجهات:

بباسطة مبدأ تعدد الواجهات Polymorphism هو قدرة الفئة على احتوائها اكثر من واجهة بحيث تمكنك من توحيد عدة فئات مختلفة باسماء اعضاء متشابهه، فلو امعنت النظر قليلاً في ادوات Visual Basic ستجد انها مختلفة المهام والانجازات الا انها تحتوي على خصائص، طرق واحداث مشتركة كـ Left، Moveو Click مما يسهل عليك كمبرمج حفظها وتوحيد الاجراءات التي تستخدم هذه الاعضاء. الفصل القادم يناقش مبدأ تعدد الواجهات بالتفصيل.

الوراثة:

الوراثة Inheritance هي قدرة فئة على اشتقاق اعضاء من فئة ام بحيث تزيد من قوة الفئة الوارثة وتضيف اعضاء جديدة للفئة الام، فلو كان لديك فئة قوية واردت اضافة طريقة او خاصية لها، فلا يوجد داعي لاعادة بناء الفئة من جديد واضافة الخاصية او الطريقة المطلوبة، فكل ما ستقوم به هي عملية انشاء فئة خالية تضيف اليها الخاصية او الطريقة التي تريدها ومن ثم تشتقها من الفئة التي تريد تطويرها واضافة الخاصية او الطريقة لها. الفصل القادم يناقش مبدأ الوراثة بالتفصيل.

بناء اول فئة مبسطة

والان شغل Visual Basic كي نبدأ بتصميم اول فئة تمثل شخص سنسميها CPerson. اختر الامر Add Class Module من قائمة Project، ومن صندوق الحوار الذي –قد- يظهر امامك، اختر الرمز Class Module وانقر على الزر Open لتظهر لك نافذة اكواد تعريف الفئة، اضغط على المفتاح [F4] لعرض نافذة خصائص الفئة، وعدل خاصية الاسم من Class1 الى CPerson، واكتب هذا الكود في الفئة:

```
Public sName As String
Public dBirthDate As Date
```

وبهذا نكون قد انجزنا اول فئة بالاسم CPerson تحتوي على الخاصيتين sName و dBirthDate. تستطيع استخدام الفئة CPerson في أي مكان داخل مشروعك، اذهب الى الحدث Click التابع لنافذة النموذج واكتب هذا الكود:

```
Private Sub Form_Click()
    Dim Turki As New cPerson
    Dim Khaled As New cPerson

    Turki.sName = "تركي العسيري"
    Turki.dBirthDate = #1/1/1900#
    Khaled.sName = "خالد الابراهيم"
    Khaled.dBirthDate = #1/1/1979#
    Print Turki.sName, Turki.dBirthDate
    Print Khaled.sName, Khaled.dBirthDate
End Sub
```

قمنا -في الكود السابق- بانشاء كائنين Turki و Khaled من الفئة التي صممناها للتو CPerson، ومن ثم قمنا بتعيين قيم للخاصيتين sName و dBirthDate لكل كائن على حدة، وختمنا الكود بطباعة قيم الخصائص التابعة للكائنين Turki و Khaled.
صحيح ان الفئة السابقة لن تطبقها في حياتك البرمجية –بشكل جاد- لطباعة قيم متغيرات، الا ان الغرض الاساسي هو مجرد توضيح فكرة الفئات وطريقة استخدامها.

بناء الفئات

والان بعد ان عرفتك على الفكرة الاساسية من الفئات Classes سنبدأ بالتوغل في تفاصيل بناء خصائصها، طرقها واحداثها حتى تزيد من قوة الفئة.

بناء الخصائص

اذا عدنا الى مثال الفئة CPerson السابق، فسنلاحظ ان المبرمج يستطيع اسناد أي قيمة للخاصية dBirthDate، وقد يعطي فرصة للمستخدم بادخال العمر من خانة نص:

```
Turki.iAge = CDate ( Text1.Text )
```

المشكلة في الكود السابق، ان المستخدم بامكانه ادخال أي عدد يمثل تاريخ ميلاد الشخص وقد يكون تاريخ لم يحل بعد، لذلك عليك التحقق من تاريخ الميلاد في كل مرة تمكن المستخدم من ادخال قيمة للخاصية dBirthDate:

```
If CDate( Text1.Text ) > Date Then
        MsgBox "خطأ في القيمة"
Else
        Turki. dBirthDate= CDate(Text1.Text)
End If
```

يعيب الكود السابق انه يلزمك بعملية التحقق من القيمة في كل مرة تريد اسناد قيمة للخاصية dBirthDate، والحل هو باستخدام نوع خاص من الاجراءات يسمى الخصائص والتي تمثل خصائص الكائن لتحميه من القيم الخاطئة:

```
Private m_dBirthDate As Date

Public Property Get dBirthDate () As Date
   dBirthDate= m_dBirthDate
End Property

Public Property Let dBirthDate (ByVal dNewValue As Date)
   If dNewValue > Date Then
      MsgBox "خطأ في القيمة"
      m_dBirthDate = Date
   Else
      m_dBirthDate= dNewValue
   End If
End Property
```

ملاحظة: في الحقيقة، اظهار رسالة MsgBox من داخل الفئة –كما في الكود السابق- يعتبر اسلوب غير احترافي وتصميم سيء جدا للفئات خاصة عندما تزيد احجامها، ويفضل ارسال رسالة خطأ بالطريقة Err.Raise بدلا من MsgBox. الا انني استخدمت الدالة MsgBox في المثال لتقريب الفكرة اليك.

سيناريو تنفيذ الاجراءات السابقة سيكون كالتالي: في كل مرة تقوم بتعيين او اسناد قيمة جديدة للخاصية dBirthDate، سيتم استدعاء الاجراء Let dBirthDate وارسال القيمة الجديدة الى المتغير dNewValue، وفي كل مرة تقوم بقراءة قيمة الخاصية dBirthDate، سيتم استدعاء الاجراء Get dBirthDate والذي يعود بقيمة الخاصية. بامكانك ايضا اضافة خاصية جديدة iAge دون الحاجة لتعريف متغير خاص Private لها:

```
Public Property Get iAge() As Integer
   iAge = DateDiff("yyyy", m_dBirthDate, Date)
End Property
```

تلاحظ انني لم استخدم الا اجراء واحد وهو Get iAge وتجاهلت الاجراء Let iAge وذلك لاني اريد ان اجعل الخاصية iAge للقراءة فقط Read Only، فلو حاول المبرمج تعيين او كتابة قيمة جديدة للخاصية ستظهر رسالة خطأ Read Only Property:

```
Print Turki.iAge        ' ممكن جدا
Turki.iAge = 80         ' رسالة خطأ
```

وبامكانك تطبيق العكس، أي استخدام الاجراء Property Let دون الاجراء Property Get لتجعل الخاصية للكتابة فقط Write Only Property:

```
Private m_sPassword As String

Public Property Let sPassword ( sNewValue As String)
   m_sPassword = sNewValue
End Property
```

المزيد ايضا، يمكنك التعامل مع اجراءات الخصائص كاجراءات Sub's او Functions عادية لتمكنها من استقبال قيم Parameters:

```
Private m_sAddress (2) As String

Public Property Get sAddress( iIndex As Integer) As String
   sAddress = m_sAddress ( iIndex )
End Property
```

```
Public Property Let sAddress(iIndex As Integer, sNewValue As String)
   m_sAddress ( iIndex ) = sNewValue
End Property
```

وبامكانك استدعاء الخاصية sAddress بهذه الطريقة:

```
Turki.sAddress (0) = "حي الوهم – شارع الحقيقة"
Turki.sAddress (1) = "ولاية فلوريدا – باكستان"
Turki.sAddress (2) = "هاتف منزل 999"
```

وعند الحديث عن الخصائص التي تمكنك من اسناد قيم لكائنات، فعليك استخدام الاجراء Property Set عوضا عن الاجراء Property Let:

```
Private m_PersonParent As CPerson

Public Property Get PersonParent( ) As CPerson
   Set PersonParent = m_PersonParent
End Property

Public Property Set PersonParent( ByVal objNewValue As CPerson )
   Set m_PersonParent =  objNewValue
End Property
```

مواصفات الخصائص Property Attributes:

بامكانك تعديل مواصفات الخصائص عن طريق صندوق الحوار Procedure Attributes والذي تصل اليه من القائمة Tools بعد تحريك مؤشر الكتابة الى مكان اجراء الخاصية. من هذه المواصفات: جعل الخاصية افتراضية Default Propertey، اخفاء الخاصية من نافذة مستعرض الكائنات Object Browser، كتابة وصف للخاصية ....الخ، بامكانك استكشاف باقي الخيارات في صندوق الحوار، وعليك ان تعلم علم اليقين ان جميع هذه الخيارات ستحفظ في ملف الفئة CLS. فقط ولن تظهر لك في نافذة محرر اكواد الفئة، فلو قمت بعملية نسخ ولصق اكواد الفئة الى فئة اخرى، عليك اعادة عملية تحرير مواصفات الفئة.

بناء الطرق

الطرق Methods ماهي الا اجراءات Sub’s او دوال Function معرفة داخل الفئـة، ولا اعتقد انك بحاجـة الـى اعـادة الفصـل الثالـث "لغـة البرمجـة BASIC" لتفاصـيل بنـاء الاجراءات والدوال. مع ذلك، هذا مثال لطريقة تابعة للفئة CPerson:

```
Public Sub SetData(sName As String, dBirthDate As Date, sAddress As Variant)
   Me.sName = sName
   Me.dBirthDate = dBirthDate
   Me.sAddress(0) = sAddress(0)
   Me.sAddress(1) = sAddress(1)
   Me.sAddress(2) = sAddress(2)
End Sub
```

بامكانك استدعاء هذه الطريقة بدلا من تعيين كل خاصية على حده:

```
' بدلا من تعيين الخصائص
Turki.sName = txtName.Text
Turki.dBirthDate = CDate ( txtBirthDate.Text )
Turki.sAddress (0) = txtAddress1.Text
Turki.sAddress (1) = txtAddress2.Text
Turki.sAddress (2) = txtAddress3.Text

' استدعي الطريقة
Turki.SetData txtName, CDate(txtBirthDate), Array(txtAddress1, _
                  txtAddress2, txtAddress3)
```

ملاحظة:	حتى لو لم تقتع بفكرة تعيين الخصائص باستخدام الطـرق كمـا فـي المثال السابق، تذكر ان استدعاء الطريقـة السـابقة اسـرع بخمـس مرات من تعيين قيمة كل خاصية على حده، وسـتؤثر هـذه السـرعة كلما كانت اكواد الكائن ابعد –كمكونات COM او DCOM.

بناء الاحداث

عندما نربط بين كلمتي الاحداث والفئات يتبادر لذهن مبرمجي VB4 حدث الانشاء Class_Initialize والانهاء Class_Terminates. لكن مع الاصدارات الاحدث، اصبحت الفئات قابلة على انشاء وتعريف احداث جديدة قابلة للتصريح من العملاء Clients المستخدمين لتلك الفئة –والذين قد تكون انت احدهم.
الفكرة ليست صعبة او مختلفة عن الاحداث الموجودة في الادوات، لنأخذ مثلا نافذة النموذج Form، تم تعريف حدث فيها باسم Click ولاشتقاق واستخدام ذلك الحدث كل ما هو مطلوب منك وضع اسم الكائن ثم شرطة سفلية ومن ثم اسم الحدث كما في هذا الكود:

```
Private Sub Form_Click()
اشتقاق حدث النقر من كائن النموذج '
End Sub
```

من هنا يتضح لنا ان الحدث بكل بساطة عبارة عن اجراء Sub قد يحتوي على متغيرات اضافية Parameters كالموجودة في حدث MouseDown او KeyPress. الذي سنفعله هنا بالضبط هو تعريف حدث باسم DataHasBeenSent موجود في الفئة CPerson. ولتعريف هذا الحدث الجديد في الفئة سنستخدم الكلمة المحجوزة Event:

```
تعريف حدث جديد '
Event DataHasBeenSent(objTo As CPerson, bSuccess As Boolean)
```

لكن مهلا! متى يستم تنفيذ هذا الحدث؟ هل سيكون ذلك عند استخدام خصائص وطرق الكائن كل مرة؟ ام عندما تزداد شهوة Visual Basic لتفجير الاحداث؟ والجواب في أي وقت تريده عن طريق استخدام العبارة RaiseEvent في داخل الفئة. اكتب هذا الكود في الفئة CPerson:

```
Option Explicit

تعريف حدث جديد '
Event DataHasBeenSent(objTo As CPerson, bSuccess As Boolean)

تعريف طريقة جديدة في الفئة '
Public Sub SendData(objTo As CPerson)
```

```
' هنا سنقوم بتفعيل الحدث
  If objTo Is Nothing Then
    RaiseEvent DataHasBeenSent(objTo, False)
  Else
    RaiseEvent DataHasBeenSent(objTo, True)
  End If
End Sub
```

والان ننتقل الى الجهة الاخرى ومعرفة كيفية التفاعل مع هـذا الحـدث كمـا تتفاعـل مع احداث الادوات الاخرى كـ Click وغيرها، تتم العملية بنفس الطريقة التي تتعامل مع الادوات شريطة:

- استخدام الكلمة المحجوزة WithEvents مع تعريف الكائن.
- لا يكون الكائن تابع لمصفوفة.
- ان يتم الاعلان عن الكائن على مستوى الوحدة -أي Public.

سننشئ كائن باسم Caller من الفئة CPerson وسنحاول الاستجابة لاحداث ذلك الكائن، في نافذة النموذج اكتب هذا الكود:

```
Option Explicit

Dim WithEvents Caller As CPerson

Private Sub Form_Click()
        Dim Khaled As CPerson

        Set Khaled = New CPerson

        Khaled.sName = "خالد"

        Caller.SendData Khaled
End Sub

Private Sub Form_Load()
  Set Caller = New MyClass
End Sub
```

```
Private Sub Caller_DataHasBeenSent(objTo As CPerson, bSuccess As Boolean)
   If bSuccess Then
      MsgBox " تم ارسال البيانات بنجاح الى: " CPerson.sName
   Else
      MsgBox "لم اتمكن من ارسال البيانات"
   End If
End Sub
```

ملاحظة:	لا يوجد داعي لكتابة اسم الحدث الطويل بلوحة المفاتيح، فبمجرد تعريفك للسطر الذي توجد به كلمة WithEvents انتقل الى القائمة المنسدلة Combo Box في اعلى يسار نافذة التحرير حتى ترى اسم الكائن MyCaller مع باقي اسماء الادوات الموجودة في النافذة.

مثال مبسط جدا يوضح طريقة الاستجابة للحدث Caller_DataHasBeenSent، الذي يتم تفجيره بمجرد نجاح الطريقة SendData.

القاء الاحداث:

عن طريقة الكلمة المحجوزة WithEvents تستطيع تطبيق مبدأ القاء الاحداث Event Multicasting، وهي عملية رمي الاحداث من كائن او اداة الى فئة كائن آخر بمجرد تفجير الحدث وقبل ان تنفيذ اكواده. سأوضح الفكرة بالمثال القديم الموجود في الفصل الثاني "النماذج والادوات" وبالتحديد عند فقرة "السيطرة على المدخلات" التابعة لفقرة "أداة النص TextBox"، تلاحظ ان الكود المستخدم للسيطرة على المدخلات كان طويل جدا، وقد اتفقنا –منذ البداية- انه من غير المعقول استخدام كل هذه الاكواد للتحقق من القيمة التي يكتبها المستخدم في خانة النص، ولكن هنا سنستخدم الكود مرة واحدة فقط، وسنضعه في فئة باسم CNumTextBox:

```
' الاداة التي ستلقي احداثها الينا
Public WithEvents TextControl As TextBox

' اكواد تمنع المستخدم من كتابة الا الارقام
Private OldText As String
Private OldSelStart As Long
```

```
Private Sub TextControl_GotFocus()
' عندما يكون التركيز على الاداة
' لابد من حفظ قيمتها
        OldText = TextControl.Text
        OldSelStart = TextControl.SelStart
End Sub

Private Sub TextControl_Change()
        ' متغير يمنع استدعاء الاجراء تراجعيا
        Static bExitNow As Boolean
        If bExitNow Then Exit Sub

        If Not IsNumeric(TextControl.Text) Then
                ' المفتاح المدخل ليس رقم
                ' قم باعادة عرض القيمة القديمة
                bExitNow = True
                TextControl.Text = OldText
                bExitNow = False
                TextControl.SelStart = OldSelStart
        Else
                ' القيمة المدخلة رقمية اذا
                ' قم بحفظها
                OldText = TextControl.Text
                OldSelStart = TextControl.SelStart
        End If
End Sub

Private Sub TextControl_KeyDown(KeyCode As Integer, Shift As Integer)
        OldSelStart = TextControl.SelStart
End Sub

Private Sub TextControl_KeyUp(KeyCode As Integer, Shift As Integer)
        OldSelStart = TextControl.SelStart
End Sub

Private Sub TextControl_MouseUp(Button As Integer, Shift As Integer, _
```

```
                X As Single, Y As Single)
        OldSelStart = TextControl.SelStart
End Sub

Private Sub TextControl_Click()
        OldSelStart = TextControl.SelStart
End Sub
```

والان في كل مرة تريد انشاء اداة نص TextBox جديدة لا تقبل الا الاعداد، فلا يوجد داعي لكتابة كل الاكواد السابقة، وانما قم بالقاء جميع احداث اداة النص الى الفئة:

```
Dim NumText As New CNumTextBox
Dim NumText2 As New CNumTextBox

Private Sub Form_Load()
  Text1 = "0"
  Text2 = "0"
  Set NumText.TextControl = Text1
  Set NumText2.TextControl = Text2
End Sub
```

تمكنا ببساطة شديدة في الكود السابق من جعل الاداتين Text1 و Text2 لا تقبلان الا اعداد بفضل القاء الاحداث Event Multicasting.

مثال تطبيقي

بامكانك تطبيق مئات الامثلة وانشاء مئات الفئات حتى تجعل حياتك اسهل، الا انني سأكتفي بتطبيق مثال بسيط جدا يتعامل مع الملفات الثنائية.

الفئة CFile:

قد تتعامل كثيرا مع الملفات الثنائية والتي تتطلب دقة في استخدام دوالها، عباراتها واوامرها، وبكل تأكيد الاخطاء الصغيرة تسبب الى تغيير هيئة الملف مما ينتج عنه شوائب واخطاء وقت التنفيذ. سنصمم فئة CFile تمكننا من تحرير الملفات الثنائية بطريقة اسهل، فبدلا من كتابة هذه الاكواد المعقدة:

```
Dim iFree File As Integer

iFreeFile = FreeFile
Open "MyFile.TXT" For Binary As #iFreeFile

' للكتابة الى الملف
Put #1, , "اسلوب اجرائي مقرف!"

' للقراءة
Dim sTemp As String

sTemp = String (18, " ")

Get #1, , sTemp

Print sTemp
```

ما رأيك بكتابة هذه الاكواد:

```
Dim MyFile As New CFile

MyFile.OpenFile "MyFile.TXT"
MyFile.PutData "اسلوب كائني جميل"
Print MyFile.GetData (16)
```

لا يقتصر الفرق بين الاسلوب الاجرائي الاول والاسلوب الكائني الثاني على اختصار عدد سطور الاكواد فقط، بل حتى في حالات نسيان كتابة الاوامر الضرورية، فمثلا تلاحظ انني لم اغلق الملف باستخدام الامر Close مما يؤدي الى احتجاز مساحة بالذاكرة، اضف الى ذلك احتجاز رقم الملف وعدم امكانية استخدامه لفتح ملف آخر. اما مع الاسلوب الكائني، فلا يوجد داعي لان اغلق الملف باستدعاء الطريقة CloseFile، لان الكائنات من النوع CFile تقوم باغلاق ملفاتها تلقائيا بمجرد موت الكائن، فهذا الكود قد اضفته في حدث التدمير للفئة CFile:

```
Private Sub Class_Terminate()
   Me.CloseFile
End Sub
```

الذي يقوم باستدعاء الطريقة CloseFile الخاصة باغلاق الملف:

```
Public Sub CloseFile()
    If Me.iFileNum Then
        Close #iFileNum
        m_iFileNum = 0
    End If
End Sub
```

ستجد الكثير من الخصائص التي قد اضفتها في الملف CFile.CLS وقد تضيف عشرات الخصائص بقدر ما يحلو لك. خذ مثلا هذه الخاصية lCursorLoc التي تحدد موقع مؤشر القراءة والكتابة من والى الملف:

```
Public Property Get lCursorLoc() As Long
    lCursorLoc = Seek(Me.iFileNum)
End Property

Public Property Let lCursorLoc(ByVal lNewValue As Long)
    Seek Me.iFileNum, lNewValue
End Property
```

هذه امثلة على استخدامها:

```
MyFile.lCursorLoc = 1                 ' بداية الملف
MyFile.lCursorLoc = MyFile.lLOF       ' نهاية الملف
```

راجع الملف Codes.ZIP حتى تحصل على الانجاز الكامل للفئة CFile.

استخدام الكائنات

تعرفت في الصفحات السابقة على الفئات والفكرة منها وكيفية بنائها، والان حان دور استخدام الفئات وانشاء الكائنات منها والتعرف على بعض التفاصيل المتعلقة بالكائنات.

عبارات وكلمات خاصة بالكائنات

من الضروري التعرف على العبارات والكلمات المحجوزة الخاصة بالكائنات حتى تستخدمها الاستخدام الامثل، نبدأ مع انشاء الكائنات باستخدام الكلمة المحجوزة New:

الكلمة المحجوزة New:

قبل استخدام الكائن عليك بكل تأكيد انشاء نسخة Instance منه، يمكنك Visual Basic من انشاء الكائنات بدالتين هما CreateObject و GetObject ولن اتحدث عنهما الا في الفصل الثاني عشر "برمجة المكونات COM 1"، والطريقة الاخرى التي تمكنك من انشاء الكائنات هي باستخدام الكلمة المحجوزة New سواء مع تصريح الكائن او مع العبارة Set:

```
Dim Turki As New CPerson
Dim Khaled As CPerson
Set Khaled = New CPerson
```

من الاشياء العجيبة جدا جدا والتي تغالط المنطق البرمجي، ان عملية انشاء الكائن باستخدام الكلمة New في نفس وقت التصريح –أي مع عبارة Dim مثل الكود السابق- لا تؤدي الى تفجير حدث الانشاء Class_Initialize التابع للكائن! ولن يتم تفجيره حتى تستخدم الكائن في اكوادك. والسبب الغريب جدا –بالنسبة لي- هو ان Visual Basic لن يقوم فعليا بانشاء الكائن حتى تذكره وتستخدمه في اكوادك، بالرغم من اننا استخدمنا الكلمة المحجوزة New عند التصريح لانشاءه!

العبارة Set:

تستخدم العبارة Set في العادة لاسناد كائن الى آخر:

```
Set Khaled = Turki
```

من الضروري استخدام العبارة Set عند اسناد الكائنات، لانك ان لم تستخدمها قد تظهر لك رسالة خطأ او حتى نتائج غير متوقعة، فلو افترضنا ان الكائنين Turki و Khaled لهما خاصية افتراضية واحدة هي sName، وقمت باسناد قيمة الكائن الاول الى الثاني دون استخدام العبارة Set:

```
Khaled = Turki
```

فانك في الحقيقة لم تسند الا قيمة خاصية الكائن الاول الافتراضية الى الخاصية الافتراضية للكائن الثاني، أي ان حقيقة الكود السابق هي:

```
Khaled.sName = Turki.sName
```

المعامل Is:

تستخدم هذا المعامل لمعرفة ما اذا كانا المتغيران يشيران الى نفس الكائن:

```
If Khaled Is Turki Then ...
```

ويمكنك ايضا معرفة ما اذا كان الكائن حي يرزق ويتبع لفئة او لا:

```
If Khaled Is Nothing Then ...
```

اما معامل المساواة فارجو ان تنسى فكرة استخدامه للتحقق من مساواة الكائنات، فالكود التالي:

```
If Khaled = Turki Then ...
```

لا يقارن الا الخصائص الافتراضية -ان وجدت- للكائنات، وكأنك كتبت:

```
If Khaled.sName = Turki.sName Then ...
```

العبارة TypeOf ... Is:

تمكنك هذه العبارة من اختبار نوع الفئة التي تمثل الكائن:

```
If TypeOf Turki Is CPerson Then ...
```

```
If TypeOf MyCtrl Is TextBox Then
        MyCtrl.Text = "…"
Else
        MyCtrl.Caption = "…"
End If
```

الدالة TypeName:

الدالة TypeName تعود بقيمة حرفية تمثل اسم الفئة التابع لها الكائن:

```
Print TypeName(Turki)        ' CPerson
Print TypeName(Text1)        ' TextBox
```

القيمة Nothing:

بامكانك الغاء الكائن في أي وقت بمجرد اسناد القيمة Nothing اليه:

```
Set Khaled = Nothing
```

المزيد من التفاصيل حول موت الكائنات ستقرأها قريبا.

ماهي حقيقة الكائن؟

في البداية اود ان اعرف ماهو الكائن -متغير الكائن ان صح التعبير- يا ترى؟ والجواب بكل بساطة: متغير الكائن عبارة عن منطقة موجودة في الذاكرة تحمل بيانات تتعلق بذلك الكائن. قد تكون اجابة السؤال مستنبطة من مبدأ تعريف التركيبات UDT الا ان الاجابة السابقة مع الاسف الشديد خاطئة 100% ! فمتغير الكائن مهما كان نوعه فان حجمه لا يزيد عن 4 بايت (في نظم 32 بت) لانه عبارة عن مؤشر الى منطقة في الذاكرة تحمل بيانات تتعلق بذلك الكائن والدليل راقب هذا الكود:

```
Dim X As New MyClass
Dim Y As New MyClass

' المؤشران
' X, Y
' يشيران الى نفس الكائن
Set Y = X
```

```
Y.Value = 100

' Y.Value = 100
Print Y.Value

X.Value = 200

' Y.Value = 200  !!
Print Y.Value
```

عندما يتم تنفيذ السطر Set Y = X فان المنطقة مـن الـذاكرة التـي كـان يشـير لها المتغيـر -المؤشـر- X اصـبحت نفـس المنطقـة التـي يشـير لهـا المتغيـر -الكـائن- Y والدليل على ذلك، انني عنـدما قمـت بتعـديل قيمـة الخاصـية X.Value = 200 فـان الخاصية Y.Value تأثرت بسبب التعديل وذلك لان X و Y متغيران (كائنـان، مؤشـران) يشيران الى نفس المنطقة من الذاكرة التي تحتوي علـى بيانـات تتعلـق بالكـائن X وليس منطقتين مختلفتين. وبكل تأكيد ستسأل نفسـك عـن المنطقـة التـي كانـت مخصصة للكائن Y ما هي اخبارها يا ترى؟ اخبارها يا قارئي العزيـز فـي المشـمش! لانها قد اختفت من الذاكرة وانتهت أي بعبارة لغوية ماتت وستعرف السبب لاحقا.

صورة الكائن في الذاكرة

الكائنات ليست كالمتغيرات العادية فهي تحجز لنفسها منطقتين في الذاكرة الاولى خاصة لمؤشر بيانـات الكـائن فـي الـذاكرة -حجمـه 4 بايـت- والثانيـة خاصـة لبيانـات الكائن نفسه. اما عندما تعلن عـن متغيـر عـادي كــ Integer او Long فـلا نحتـاج الا لمنطقة واحدة بالذاكرة خاصـة بقيمـة المتغيـر باسـتثناء المتغيـرات مـن نـوع String فهي تحتاج الى منطقتين من الذاكرة مثل الكائنات.
المنطقة الثانية التي يحتجزها الكائن تعرف بمنطقة نسخة البيانات Instance Data Area والمقسمة الى ثلاثة اقسـام: قسـم خـاص بالمؤشـر VTable، وقسـم خـاص بالعداد Counter، وقسم خـاص بـالمتغيرات التابعـة للكـائن Data Area. وفيمـا يلـي تفاصيل هذه الاقسام:

المؤشر VTable:

تفاصيل هذا المؤشر لاتهم مبرمجـي Visual Basic بقـدر مـا تهـم مبرمجـي COM و C++، لكن ما استطيع قوله هو ان VTable عبارة عن مؤشر الى تركيب في منطقة اخرى بالذاكرة تمثل مواقع تنفيذ الاجراءات -الطرق Methods- وبداية كل اجراء حتى

يتم تنفيذ اكواده. كل مؤشرات VTable التابعة للكائنات تشير الى نفس التركيب في حالة كون الكائنات من نفس الفئة. فلكل فئة جدول VTable مستقل خاص باجراءات تلك الفئة، ولكل كائن مؤشر خاص به يشير الى التركيب VTable المتوافق مع الفئة المنشأ منها، فهنا:

```
Dim X As MyClass, Y As MyClass, Z As YourClass
```

توجد ثلاث مؤشرات VTable خاصة للكائنات X و Y و Z تشير الى تركيبين VTable خاصين للفئتين MyClass و YourClass.

العداد Counter:

القسم الثاني من هذه المنطقة حجمه 4 بايت وهو عبارة عن عداد يمثل عدد المؤشرات التي تشير الى هذه المنطقة. يبدأ العداد بالقيمة واحد عندما تنشئ الكائن، ويزيد كلما وجد مؤشر اخر يشير الى ذلك العداد. عندما يصل العداد الى الصفر (أي لا يوجد مؤشر يشير الى تلك المنطقة) فان المنطقة يتم تحريرها من الذاكرة وتختفي، وهذا جواب واضح للسؤال متى يموت الكائن؟ راقب هنا:

```
Dim X As MyClass, Y As MyClass

' العداد يبدأ بواحد
Set X = New MyClass
' والان العداد باثنان بسبب
' وجود مؤشران يشيران الى
' نفس المنطقة
Set Y = X
' العداد الان ينقص بواحد
Set X = Nothing
' العداد الان بصفر مما يؤدي الى
' موت الكائن
Set Y = Nothing
```

منطقة البيانات Data Area:

وهي المنطقة التي تحتوي على جميع المتغيرات العامة Public والستاتيكية Static الخاصة بالكائن، بكل تأكيد يختلف حجمها من كائن لآخر بالاعتماد على عدد

وحجم المتغيرات التابعة له. فمثلا، اذا احتوى الكائن على متغيـرين مـن نـوع Long، فان حجم هذا القسـم هو 8 بايت.

مثال توضيحي:

اود ان اوضح الاقسـام الثلاثة بمثال مـع شـكل توضـيحي لـه. بـافتراض ان لـدينا فئـة باسـم MyClass تحتوي علـى طـريقتين MyMethod1 و MyMethod2 بالاضـافة الـى متغيرين عامين Public باسـم Value1 و Value2. فالكود التالي:

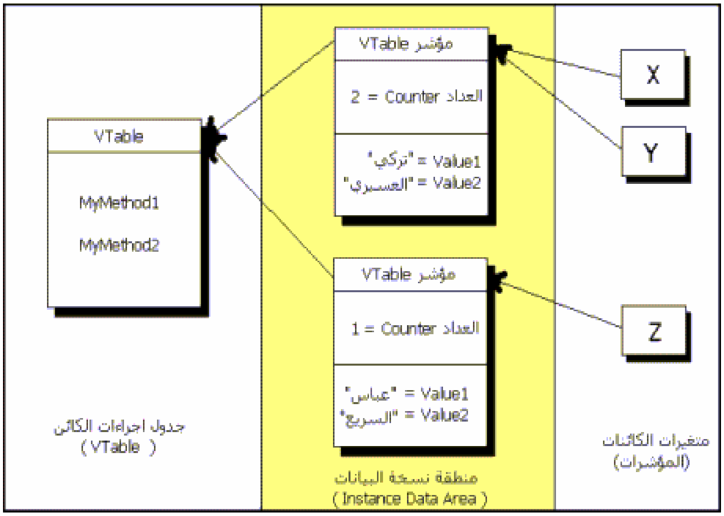
```
Dim X As MyClass, Y As MyClass, Z As MyClass

' حجز وانشاء نسخ للكائنات في الذاكرة
Set X = New MyClass
Set Y = X
Set Z = New MyClass

' تذكر ان كلاهما مؤشران لمنطقة واحدة
' X, Y
X.Value1  = "تركي"
X.Value2  = "العسيري"

' اخيرا تعيين قيم للكائن
' Z
Z.Value1  = "عباس"
Z.Value2  = "السريع"
```

يمكن ان نوضح المنطقة الخاصة بالكائنات كما في **الشكل 5-1**:



شكل 5-1: رسم توضيحي لصورة الكائنات X، Y و Z بالذاكرة.

قد لا تهمك كثيرا المربعات الموجودة في الشكل 5-1، ولكن استيعابها مسألة ضرورية لتعريف فكرة الربط Binding –عنوان الفقرة التالية- او حتى الاحتراف في برمجة مكونات COM وهو ما سنتطرق إليه في الفصول اللاحقة.

الربط Binding

عملية الربط هي باختصار ربط اعضاء Members الكائن سواء كانت خصائص او طرق بالمؤشر الذي يمثل الكائن وتحديد مواقع الاجراءات في الجدول VTable. من المعروف ان الطرق موجودة في مواقع في الذاكرة في الجدول VTable لكن عملية الوصول لها ليست مباشرة احيانا، راقب هنا:

```
Dim X As Object
If Y = True Then
  Set X = New MyClass
Else
  Set X = New YourClass
End
X.MyMethod
```

في السطر الاخير قمت باستدعاء الطريقة MyMethod التابعة للكائن X، ولكن ماهو الكائن X؟ هل هو تابع للفئة MyClass ام الفئة YourClass؟ لا نستطيع معرفة ذلك الا في وقت التنفيذ حتى تتضح الامور لـ Visual Basic، لانه يحتاج الى معرفة تركيب الجدول VTable ما اذا كان يوجد به دعم للطريقة MyMethod او لا،والتي بدورها ستأخذ وقت اطول بكثير من الوصول الى اجراء لكائن معرف النوع سابقا. لذلك، اتكلم عن نوعين من انواع الربط هما:

الربط المبكر Early Binding:

عملية الربط المبكر تتم في وقت الترجمة Compiling time والتي تقوم بتحديد مواصفات التركيب VTable مما يؤدي الى وصول اسرع بكثير لاعضاء الكائن. طبعا لعمل ذلك، لابد من ان تحدد بوضوح نوع الفئة التي سيمثلها الكائن.

```
' التصريح الواضح لنوع الكائنات
Dim X As MyClass, Y As YourClass

Set X = New MyClass

' بافتراض وجود اتصال
' COM
' طبعا ابطأ لكن يعتبر ربط مبكر ايضا
Set Y = CreateObject ("YourServer.YourClass")
```

الربط المتأخر Late Binding:

هنا يتم تحديد مواصفات التركيب VTable في وقت التنفيذ في كل مرة تصرح فيها عن كائن جديد، مما يؤدي الى بطء في تحديد الجدول VTable المناسب للكائن، والتحقق من وجود الطرق المستدعاه. المتغيرات المعرفة من نوع Object او Variant هي متغيرات لن تستطيع ان تربطها الا عن طريق الربط المتأخر.

```
' تصريح غير واضح للكائنات
Dim X As Object, Y As Variant
Set X = New MyClass

' بافتراض وجود اتصال
' COM
Set Y = CreateObject ("MyServer.YourClass")
```

ولادة وموت الكائن

ولادة الكائن هي اللحظة التي تشئ الكائن بها باستخدام الكلمة المحجوزة New او الدوال الاخرى التي لم اتطرق لها CreateObject و GetObject، ويموت الكائن بمجرد تحرير المنطقة الخاصة به في الذاكرة كخروجه عن مجاله او تعيين القيمة Nothing له. اعرض لك بعض التفاصيل الدقيقة والخاصة عن انشاء وانهاء الكائن:

انشاء الكائن واستخدامه:

عندما تقوم بانشاء الكائن لاول مرة يقوم Visual Basic بحجز منطقة في الذاكرة تمثل التركيب VTable للفئة التي تمثل ذلك الكائن. بعد ذلك، يقوم Visual Basic بحجز منطقة اخرى بالذاكرة خاصة بمنطقة نسخة البيانات Instancing Data Area والتي يقوم بتقسيمها الى ثلاثة اقسام ومن ثم تعبئة المعلومات المطلوبة في مكانها المناسب في كل قسم. القسم الاول لوضع عنوان VTable للمؤشر VTable والثاني يبدأ عداده. ولا يبدأ في القسم الاخير الا بعد تفجير الحدث Class_Initialize لان ذلك الاجراء اصبح عنوانه معروف بفضل تعريف المؤشر VTable.

اما في حالة انشاء كائن مرة اخرى، فان العملية تتم بشكل اسرع وذلك بسبب ان Visual Basic لا يقوم بحجز المنطقة بالذاكرة والخاصة بـ VTable فهي موجودة وجاهزة لاي كائن جديد سينشأ من نفس نوع الفئة السابق. اخيرا، تستطيع استخدام الكائن واستدعاء طرقه و تعيين خصائصه وحتى انتظار احداثه.

نهاية وجود الكائن بالذاكرة:

عندما يصل العداد الى صفر سيقوم Visual Basic بتفجير الحدث Class_Terminate والخاص بالكائن متيحا لك فرصة اخيرة لعمل أي شئ قبل موت الكائن ومن ثم يقوم بتحرير منطقة نسخة البيانات Instancing Data Area من الذاكرة وفقط، أي لا تتوقع ان يقوم Visual Basic بتحرير منطقة التركيب VTable لانها ستكون بالذاكرة حتى نهاية البرنامج بعبارة End واعتقد ان هذا سبب واضح في كون عملية انشاء الكائن مرة اخرى اسرع بكثير من المرة الاولى بسبب عدم ضرورة انشاء الـ VTable من جديد.

لدي نقطة اخرى حول الحدث Class_Terminate. ففي هذا الحدث تستطيع فعل ما تريد قبل موت الكائن لكن من المهم معرفة انك لن تستطيع اعادة حياة الكائن عن طريق هذا الحدث. الفكرة ببساطة هي كالانسان عندما يحتضر، فان من رحمه الله ييسر له الشهادة وقت الاحتضار وينطق بها ومن ثم يموت لكنه لن يستطيع العودة الى الحياة من جديد -الا بمعجزة الخالق الذي يحيي ويميت بكل تأكيد- اما مع Visual Basic فيوفر لك الحدث Class_Terminate فرصة اخيرة لعمل ما تريد قبل ان يموت الكائن لكنك لن تستطيع اعطائه الحياة من جديد.

نقطة اخرى حول موت الكائنات -نسأل الله طولة العمر- هـي ان الكائنـات لهـا نظـام يمنع موت الكائن اذا ما كان احد اجراءاته قيد التنفيذ. وبمعنى آخر، لنفتـرض ان احـد اجراءات الكائن يقـوم بقتـل نفسـه Set X = Nothing (اكيـد المتغيـر X مؤشـر عـام) فلابد ان تعلم ان Visual Basic راقي جدا جدا ويعلم كيف يتعامل مع هذه النوع مـن الكائنات!، فسيقوم باسلوب مهذب جدا اعطاء فرصة للكائن حتى ينهي اجراءه الذي يتم تنفيذه ومن ثم يقوم بتفجير الحدث Class_Terminate وقتل الكائن. احسـن الله عزاكم.

ارسال الكائن بالمرجع او بالقيمة

تحدثت في الفصل الثالث "لغـة البرمجـة BASIC" وبالتحديـد فـي قسـم الاجـراءات والدوال عن الفرق بين ارسال مرجع المتغير الى الاجراء وارسال قيمـة المتغيـر الـى الاجراء، وذكرت بان المتغيرات المرسلة بالمرجع يمكن لـك التعـديل فـي قيمهـا مـن نفس الاجراء، ولكن عند الحديث عن الكائنـات فحـاول نسـيان الفـرق بـين الارسـال بـالمرجع والقيمـة، لان الكـائن فـي كـلا الحـالين سيرسـل مؤشـر Pointer الكـائن وسيتمكن الاجراء من تعديل جميع محتويات الكائن.
اما الفرق بـين اسـتخدام الكلمـة المحجـوزة ByVal والكلمـة المحجـوزة ByRef فهـو فرق تقني بحت، اذ ان ارسال الكـائن بالكلمـة المحجـوزة ByVal يـؤدي الـى انشـاء نسخة جديدة من المؤشر تؤدي الـى زيـادة العـداد Counter التـابع لمنطقـة بيانـات الكائن، اما الارسال بالكلمة المحجوزة ByRef فان المؤشر هو نفـس المؤشـر الـذي ارسل الى الاجراء، هذا الكود قد يوضح الفرق:

```
Sub MySub ( objPerson As Person )  ' الارسال بالمرجع

        Set objPerson = Nothing       ' تؤدي الى موت الكائن المرسل

End Sub

Sub MySub ( ByVal objPerson As Person )   ' الارسال بالقيمة

        Set objPerson = Nothing       ' لا تؤدي الى موت الكائن المرسل

End Sub
```

الفصل السادس
تعدد الواجهات والوراثة

تحدثت في الفصل السابق عن الفئات والكائنات، وذكرت انه كلما كانت الفئة مستقلة كلما زادت امكانية اعادة استخدامها في تطبيقات اخرى، الا انك في حالات كثيرة تود توزيع الاكواد بين عدة فئات وتحاول تطبيق روابط بين الفئات لتوفر عليك عناء اعادة كتابة الاكواد المتكررة وتسهيل حياتك البرمجية بشكل افضل. في هذا الفصل سأتطرق الى مواضيع متقدمة في البرمجة كائنية التوجه واتحدث عن مبدأ تعدد الواجهات ومبدأ الوراثة، واختم الفصل بالتحدث عن فكرة الاهرام الكائنية.

تعدد الواجهات

مبدأ تعدد الواجهات Polymorphism من المبادئ التي لابد من توفرها في أي لغة برمجة كائنية التوجه OOP. ومن حسن الحظ Visual Basic يدعم الفكرة الاساسية من هذا المبدأ، سأشرح في هذه الفقرة طريقة تطبيق مبدأ تعدد الوجهات كما اتطرق الى الفئات المجردة Abstract Classes.

التعريف البرمجي لمبدأ تعدد الواجهات هو: اسماء متشابهة لكن انجازات مختلفة Same names but different implementations. المقصد من ذلك، اننا نستطيع استدعاء طرق وخصائص متشابهة الاسم لفئات مختلفة البنيان أي بانجازات مختلفة. مثال، نفترض ان لدينا فئتين الاولى CPerson والثانية CCar، كلا الفئتين يوجد بهما طريقة خاصة بالتحريك تسمى Move، وبالتالي نستطيع استدعاء الطريقتين باسمائهما: CPerson.Move و CCar.Move. لكن القضية هنا ان عملية انجاز الطريقة مختلفة رغم تشابه اسمائها، فمن المعروف ان الشخص يتحرك عن طريق قدميه اما السيارة فبلا شك تتحرك عن طريق الاربع عجلات بها، وهذا هو مبدأ تعدد الوجهات.

المزايا التي تجدها من تعدد الواجهات كثيرة ولعل الميزة الحقيقية هي انها تختصر عليك الكثير من مئات جمل الشرط كـ Select Case وغيرها. فقد تلاحظ ان مبدأ تعدد الواجهات مطبق في الادوات التي تضعها على نافذة النموذج وذلك بسبب وجود الكثير من الخصائص المشتركة بين الادوات كخاصية Left او Name وغيرها،

فلو طلبت منك احد الايام كتابة اجراء يقوم بمحاذاة اداة الـنص TextBox فـي وسـط النافذة، فستكون حصيلة اصابعك الناعمة الكود التالي:

```
CenterTextBox ( txtTextBox As TextBox )
txtTextBox.Move ScaleWidth - txtTextBox.Width) / 2, _
            ScaleHeight - txtTextBox.Height) / 2
End Sub
```

ولو كانت علاقتنا حميمـة جـدا وطلبـت منـك اجـراء آخـر يقـوم بمحـاذاة اداة العنـوان Label، فاعتقد انك ستكتب الاجراء التالي:

```
CenterLabel ( lblLabel As Label)
lblLabel.Move ScaleWidth - lblLabel.Width) / 2, _
            ScaleHeight - lblLabel.Height) / 2
End Sub
```

ولا اعتقد انك على اسـتعداد لكتابـة 18 اجـراء اخـر لتوسـيط الادوات الثمانيـة عشـر الاخرى حتى لو كانت علاقتنا عاطفية! بل ستكون مبرمج كائني التوجه اكثر وتكتـب اجراء واحد يمكن ان يستقبل أي اداة مهما كان نوعها:

```
CenterControl ( ctrlControl As Control )
ctrlControl.Move ScaleWidth - ctrlControl.Width) / 2, _
            ScaleHeight - ctrlControl.Height) / 2
End Sub
```

من الاجراء السـابق يتضـح لنـا جمـال، قـوة، ابـداع، مرونـة، فـن، وسـحر مبـدأ تعـدد الواجهات فالفئة TextBox لها واجهـة اخـرى باسـم Control تحتـوي علـى الطريقـة Move حالها كحال جميع الادوات الاخرى. الواجهة Control هي عبارة عن فئة لكنها لا تحتـوي علـى ايـة اكـواد، لـذلك تسـمى فئـة المجـردة Abstract Class Control وتحتوي على واجهة Interface، فحتى تستطيع ان تحقق مبدأ تعدد الوجهـات لابـد من وجود فئة مجردة والتي تعرف الواجهة للفئات الاخرى منها.

تطبيق عملي:

والان لنبدأ بالتطبيق، سننشـئ فئة مجردة (واجهة) باسـم ITrip والتي تمثـل رحلـة وتعريف طريقة بها لمعرفة التكاليف:

```
Function GetCost(iDistance As Integer) As Integer
' لا تكتب شيئا هنا فهذه مجرد واجهه
End Function
```

ملاحظة: جرى العرف عند مبرمجي OOP بتمييز الواجهة عن الفئة عن طريق اضافة حرف البادئة I قبل اسم الفئة، اما الفئات فما زال حرف البادئة C هو الاكثر شعبية.

والان انشاء فئة اخرى وهي تمثل رحلة بالسيارة لا تنسى ان تسميها بـ CCar:

```
' لا بد ان تضيف هذه العبارة حتى
' نستخدم الواجهة التابعة لفئة
' ITrip
Implements ITrip

Private Function ITrip_GetCost(iDistance As Integer) As Integer
' هذه الدالة مأخوذة من واجهة
' ITrip
  ITrip_GetCost = iDistance * 15
End Function
```

ملاحظة: لا يوجد داعي من كتابة الاجراء السابق بنفسك، فبمجرد كتابة الكلمة المحجوزة Implements بامكانك الوصول الى كافة اجراءات الواجهة عن طريق الاداة ComboBox الموجودة في اعلى نافذة محرر الاكواد.

ايضا فئة اخرى تمثل رحلة بالطائرة CPlane:

```
' لا بد ان تضيف هذه العبارة حتى
' نستخدم الواجهة التابعة لفئة
' ITrip
Implements ITrip

Private Function ITrip_GetCost(iDistance As Integer) As Integer
' هذه الدالة مأخوذة من واجهة
```

```
' ITrip
  ITrip_GetCost = iDistance * 100 ' لاحظ زيادة سعر التكلفة هنا
End Function
```

انتهينا من تصميم الواجهة ITrip والفئات CCar و CPlane، ولاستخدامها انتقل الى نافذة النموذج ثم ضع اداة زر اوامر واكتب هذا الكود:

```
Private Sub Command1_Click()
Dim NewTrip As ITrip

  Set NewTrip = New CCar ' الرحلة الان بالسيارة
  Print NewTrip.GetCost(50)

  Set NewTrip = New CPlane ' اصبحت بالطائرة
  Print NewTrip.GetCost(50)
End Sub
```

ستلاحظ اختلاف التكاليف بين رحلة بالسيارة واخرى بالطائرة ولو كانت المسافة متشابهه (50 كيلو متر).

الوراثة

الوراثة Inheritance هي قدرة الفئة –الفئة المشتقة Derived- على اشتقاق اعضاء فئة اخرى -الفئة الام Base Class- بحيث تتمكن الفئة المشتقة من الوصول الى جميع اعضاء (طرق/خصائص) الفئة الام، مما يؤدي الى تطوير الفئة الام واكمال نواقصها. فمثلا لو كان لدينا الفئة س ونريد اضافة الخاصية ص فيها، فلا يوجد داعي من اعادة بناء الفئة س من جديد، وانما ننشئ فئة ع تحتوي على الخاصية ص وتكون الفئة ع مشتقة من الفئة س بحيث تمتلك كافة خصائصها الاخرى. لتوضيح الفكرة، افترض ان لدينا هذه الفئات الثلاث:

1) اسم الفئة: CPerson، خصائصها: Name و Age، طرقها: Move

2) اسم الفئة: CStudent، خصائصها: Major، طرقها: ChangeCollege

3) اسم الفئة: CWorkman، خصائصها: Salary، طرقها: ChangeDepartment

سنطبق عليها مبدأ الوراثة الان بجعل الفئتين CWorkman و CStudent مشتقة ووارثة لاعضاء الفئة الام CPerson. أي ان الفئتان CStudent و CWorkman قابلتان للوصول الى اعضاء الفئة الام CPerson، لذلك جميع هذه الاكواد صحيحة من الناحية المنطقية:

```
CStudent.Name = "محمد"
CStudent.Age = 25
CStudent.Move()
CStudent.Major = "علوم الحاسب"
CStudent.ChangeCollege()

CStudent.Name = "عبدالله"
CWorkman.Age = 30
CWorkman.Move()
CWorkman.Salary = 10,000
CWorkman.ChangeDepartment()
```

والسبب في ذلك، ان الفئات المشتقة CWorkman و CStudent قابلة للوصول الى جميع عناصر الفئة الام CPerson، والعكس غير صحيح! فالفئة الام CPerson لا تستطيع الوصول الى اعضاء الفئة المشتقة منها، فلا تكتب في احد الايام شيئا من هذا القبيل:

```
CPerson.Salary = 20,000
' او
CPerson.ChangeCollege()
```

المزيد ايضا، الفئات المشتقة ترث من الفئات المشتق منها (الفئات الام) فقط. ففي مثالنا السابق، الفئات CStudent و CWorkman لا يمكن لاي فئة منها الوصول الى اعضاء الفئة الاخرى فلا تكتب مثل هذا:

```
CStudent.Salary = 10,000
CWorkman.ChangeCollege()
```

لانهما مشتقان من الفئة الام CPerson فقط. وهذا باختصار مفهوم مبدأ الوراثة في جميع لغات البرمجة والذي يقدم لك الكثير من اختصار كتابة الاكواد والتسهيل في

عملية التنقيح وتطوير الفئة نفسها ايضا، تخيل مثلا ان لديك فئة MyClass واردت تطويرها باضافة عناصر جديدة لها، كل ذلك يمكن ان يتم عن طريق اشتقاق فئة اخرى جديدة منها واضافة اللازم.

محاكاة الوراثة بـ Visual Basic

للاسف الشديد Visual Basic لا يدعم مبدأ الوراثة بشكل ضمني، والذي سنفعله هنا عملية محاكاة مبدأ الوراثة على الفئات. الفكرة في محاكاة الوراثة سهلة، فبما ان الفئات المشتقة سترث نفس اكواد الفئات الام، فلماذا لا نقوم بنسخ جميع محتويات الفئة الام ولصقها في الفئات المشتقة. لتطبيق ذلك، انشئ فئة باسم CPerson واكتب فيها هذا الكود:

```
Private m_sName As String
Private m_iAge As Integer

Sub Move()
  MsgBox "تم تنفيذ اجراء التحريك"
End Sub

Property Get iAge () As Integer
  iAge = m_iAge
End Property

Property Let iAge ( iNewValue As Integer )
  m_iAge = iNewValue
End Property

Property Get sName () As String
  sName = m_sName
End Property

Property Let sName ( sNewValue As String  )
  m_sName = sNewValue
End Property
```

ولانشاء فئة الـ CStudent، قم بنسخ جميع محتويات الفئة الام CPerson ولصقها في الفئة المشتقة:

```
' خصائص الفئة الام
Private m_sName As String
Private m_iAge As Integer

Property Get iAge () As Integer
  iAge = m_iAge
End Property

Property Let iAge ( iNewValue As Integer )
  m_iAge = iNewValue
End Property

Property Get sName () As String
  sName = m_sName
End Property

Property Let sName ( sNewValue As String )
  m_sName = sNewValue
End Property

' خصائص الفئة المشتقة
Private m_sMajor As String

Property Get sMajor () As String
  sName = m_sName
End Property

Property Let sMajor ( sNewValue As String )
  m_sName = sNewValue
End Property

' طرق الفئة الام
Sub Move()
  MsgBox "تم تنفيذ اجراء التحريك"
```

```
End Sub

' طرق الفئة المشتقة
Sub ChangeCollege()
  MsgBox "تم تنفيذ اجراء تحويل الكلية"
End Sub
```

نفس الفكرة طبقها على الفئة CWorkman:

```
' خصائص الفئة الام
Private m_sName As String
Private m_iAge As Integer

Property Get iAge () As Integer
  iAge = m_iAge
End Property

Property Let iAge ( iNewValue As Integer )
  m_iAge = iNewValue
End Property

Property Get sName () As String
  sName = m_sName
End Property

Property Let sName ( sNewValue As String )
  m_sName = sNewValue
End Property

' خصائص الفئة المشتقة
Private m_lSalary As Long

Property Get lSalary () As Long
  lSalary = m_lSalary
End Property

Property Let lSalary ( lNewValue As Long )
```

```
  m_lSalary = lNewValue
End Property

' طرق الفئة الام
Sub Move()
  MsgBox "تم تنفيذ اجراء التحريك"
End Sub

' طرق الفئة المشتقة
Sub ChangeDepartment()
  MsgBox "تم تنفيذ اجراء تغيير القسم"
End Sub
```

والان قمنا بعملية محاكاة مبدأ الوراثة، فتستطيع كتابة اكواد مثل:

```
Dim X As New CStudent
Dim Y As New CWorkman

X.sName  = "محمد"
X.iAge = 25
X.Move()
X.sMajor = "علوم الحاسب"
X.ChangeCollege()

Y.sName = "عبدالله"
Y.iAge = 30
Y.Move()
Y.lSalary = 10,000
Y.ChangeDepartment()
```

علاقة "يحتوي على"

المشكلة في فكرة المحاكاة السابقة هي ضرورة وجود الشيفرة المصدرية Source Code للفئة الام حتى تتمكن من اشتقاق الفئات منها. اما في حالة كون الفئة في ملف تنفيذي -كداخل مكون COM مثلا- فالعملية معقدة جدا -ان لم تكن مستحيلة. والحل عن طريق تطبيق علاقة تعرف في عالم البرمجة كائنية التوجه OOP باسم يحتوي على Has a وهي تنص باختصار على ان الفئة يمكن لها ان تحتوي على

كائن من فئة اخرى عن طريق تعريف متغير يمثل كائن لتلـك الفئـة. الان قـم باعـادة تصميم الفئات المشـتقة CStudent و CWorkman بهذه الطريقة:

```
' الفئة الام
Public objPerson As New CPerson

' خصائص الفئة المشـتقة
Private m_sMajor As String

Property Get sMajor () As String
   sName = m_sName
End Property

Property Let sMajor ( sNewValue As String )
   m_sName = sNewValue
End Property

' طرق الفئة المشـتقة
Sub ChangeCollege()
   MsgBox "تم تنفيذ اجراء تحويل الكلية"
End Sub
```

العيب الوحيد في هذه الطريقة هو ان المسـتخدم لهذه الفئـة لـن يسـتطيع محاكـاة الوراثة بشـكلها الصحيح، فلـن يسـتطيع كتابـة العبـارة مـثلا X.sName للوصـول الـى اعضاء الفئة الام، وانما سـيضطر الى اسـتخدام الكائن المحضون في الفئة المشـتقة وكتابة X.objPerson.sName.

التفويض Delegation

يبدو ان الحل الامثل هو بجعل كائن الفئة الام مخفـي ومحاكـاة جميـع اعضـاءه فـي الفئة المشـتقة، ومن ثم ارسالها الى الكائن، وهذه هي الفكرة الاسـاسـية من مبـدأ التفويض، فسـيصبح الكود النهائي للفئة المشـتقة CStudent:

```
' الفئة الام
Private objPerson As New CPerson

' تفويض خصائص الفئة الام
```

```
Property Get iAge () As Integer
  iAge = objPerson.iAge
End Property

Property Let iAge ( iNewValue As Integer )
  objPerson.iAge = iNewValue
End Property

Property Get sName () As String
  sName = objPerson.sName
End Property

Property Let sName ( sNewValue As String )
  objPerson.sName = sNewValue
End Property

' خصائص الفئة المشتقة
Private m_sMajor As String

Property Get sMajor () As String
  sName = m_sName
End Property

Property Let sMajor ( sNewValue As String )
  m_sName = sNewValue
End Property

' تفويض طرق الفئة الام
Sub Move()
  objPerson.Move
End Sub

' طرق الفئة المشتقة
Sub ChangeCollege()
  MsgBox "تم تنفيذ اجراء تحويل الكلية"
End Sub
```

والان قمت بمحاكاة مبدأ الوراثة في Visual Basic لكـن -مـع الاسـف- اود ان اخبـرك ان الذي فعلناه صحيح 100% في هذا المثال فقط! لانـه فـي حالـة كـون للفئـة الام واجهة فرعية من واجهة اخرى -مبدأ تعدد الواجهـات Polymorphism- لـن تسـتطيع الوصول الى اعضاء الواجهة الاخرى للفئة، والحل تجده في الفقرة التالية.

وراثة الواجهات

اريد ان ابدأ هنا تقديم محتويات المثـال الـذي سـيظهر لنـا مشـكلة وراثـة الواجهـات وكيفية تلافيها، سيكون لدينا في هذا المثال واجهـة واحـدة باسـم IMyInterface و فئة ام باسـم CBaseClass وفئة مشتقة باسـم CDerivedClass.

الواجهة IMyInterface تحتوي علـى اجـراء باسـم MyMethod، والفئـة الام تحتـوي على اجراء باسم BaseMethod. واخيرا، الفئة المشتقة تحتـوي علـى اجـراء باسـم DerivedMethod، ضع فـي عـين الاعتبـار علـى ان الفئـة الام تحتـوي علـى واجهـة اضافية من الواجهة IMyInterface أي ان الطرق التابعة للكائن مـن الفئـة الام هـي MyMethod و BaseMethod.

توضيح المشكلة:

حتى نقوم بحل المشكلة لابد بكل تأكيد من معرفة ماهي المشـكلة. الان سـنقوم بعمليـة التفـويض -لمحاكـاة الوراثـة- كمـا عملنـا فـي الفقـرة السـابقة وجعـل الفئـة CDerivedClass مشتقة مـن الفئـة الام CBaseClass، ليصـبح الكـود النهـائي للفئـة CDerivedClass هو:

```
Private BaseClass As New CBaseClass

' تفويض طرق الفئة الام
Sub BaseMethod()
   BaseClass.BaseMethod
End Sub

' طرق الفئة المشتقة
Sub DerivedMethod()
       ' اكتب ما تريده هنا
End Sub
```

يبـدو ان المشـكلة اتضـحت لـك الان وهـي ان الفئـة المشـتقة لا تـدعم الطريقـة MyMethod التابعة للواجهـة IMyInterface والتـي تعتبـر احـدى واجهـات الفئـة الام CBaseClass. اذا كنت تفكر بتفويض اجراء لعمل ذلك كهذا:

```
Sub MyMethod()
  BaseClass.MyMethod
End Sub
```

فارجو ان توقف القراءة في الحال! لانك بحاجـة ماسـة الـى معرفـة واسـتيعاب مبـدأ تعدد الواجهات Polymorphism، فالكـائن BaseClass معـرف مـن الفئـة CBaseClass وليس من الواجهة IMyInterface ولن تسـتطيع عمل ذلك.

حل المشكلة:

اتمنى ان تكـون المشـكلة قـد اتضـحت لـك، يكمـن الحـل بـالالتزام بعمليـة تضـمين الواجهة IMyInterface في الفئة المشـتقة، ويصبح الكود النهائي بهذا الشـكل:

```
Private BaseClass As New CBaseClass
' لابد من تضمين تلك الواجهة
Implements IMyInterface

' تفويض طرق الفئة الام
Sub BaseMethod()
  BaseClass.BaseMethod
End Sub

' تفويض الواجهة الاخرى للفئة الام
Private Sub IMyInterface_MyMethod()
  Dim TempInf As IMyInterface

  Set TempInf = BaseClass
  TempInf.MyMethod ' الان يمكنك عمل ذلك
End Sub

Sub MyMethod()
  IMyInterface_MyMethod
End Sub
```

```
` طرق الفئة المشتقة
Sub DerivedMethod()
` اكتب ما تريده هنا
End Sub
```

التصنيف الفرعي Subclassing

عملية التفويض Delegation بلا شك غير مقبولة بشكل كبير لدى مبرمجي OOP وبالتحديد مستخدموا مبدأ الوراثة بكثرة، الا انها تتميز باعطاءك تحكم اكبر قبل تنفيذ الاجراء التابع للفئة الام من داخل الفئة المشتقة. فمثلا، الفئة المشتقة تحتوي على هذا الكود:

```
Sub BaseMethod()
  BaseClass.BaseMethod
End Sub
```

نفترض ان الطريقة السابقة عبارة عن دالة Function وتحتوي على متغيرات Parameters:

```
Function BaseMethod (X As Long, Y As Long) As Long
  BaseMethod = BaseClass.BaseMethod (X, Y)
End Sub
```

الذي كنت اقصده من التحكم الاكبر هو انك في الكود السابق تستطيع الغاء عملية استدعاء الطريقة الموجودة في الفئة الام او تعديل قيم المتغيرات المرسلة Arguments او حتى تغيير القيمة التي تعود بها الطريقة Returned Value، فتستطيع ان تكتب شيئا مثل:

```
Function BaseMethod (X As Long, Y As Long) As Long
  If X = 0 Then
    BaseMethod = 0
  ElseIf Y = 0 Then
    BaseMethod = BaseClass.BaseMethod (X, 1)
  Else
    BaseMethod = BaseClass.BaseMethod (X, Y)
```

```
  End If
End Sub
```

في مبدأ الوراثة، تسمى هذه العملية بالتصنيف الفرعي للفئة الام Subclassing the base class، وهي من التقنيات المتقدمة التي توفرها لك لغات البرمجة كائنية التوجه OOP كلغة الـ ++C والتي طبقناها بشكل فعال في لغتنا الجميلة Visual Basic.

المزيد ايضا، يمكنك تطبيق مبدأ التصنيف الفرعي على جميع اعضاء مكتبات VB و VBA، فمثلا الدالة Hex حروف الاعداد الستعشرية التي تعود بها تكون انجليزية دائما A، B، .... F وقد تكون من المتعصبين الى لغتنا الجميلة بحيث تود ان تعود الدالة بالحروف العربية، فتستطيع تطبيق مبدأ التصنيف الفرعي للدالة Hex بتعريف هذه الدالة في ملف برمجة BAS:

```
Function Hex(lNum As Long) As String
   Hex = VBA.Hex$(lNum)
   Hex = Replace(Hex, "A", "ا")
   Hex = Replace(Hex, "B", "ب")
   Hex = Replace(Hex, "C", "ت")
   Hex = Replace(Hex, "D", "ث")
   Hex = Replace(Hex, "E", "ج")
   Hex = Replace(Hex, "F", "ح")
End Function
```

يمكنك استدعائها بنفس الطريقة:

```
Dim lCounter As Long

For lCounter = 0 To 15
   Print Hex(lCounter)
Next
```

الاهرام الكائنية

عندما تصبح مبرمج كائني التوجه، فان نظرتك الى عملية بناء وتصميم البرنامج تنتطلق من محور كائناته وليس اكواده، مجموعة الكائنات التي تصممها تسمى

الاهرام الكائنية Object Hierarchies الخاصة ببرنامجك. فلو تنظر الى معظم البرامج التجارية كـ Microsoft Word، Microsoft Excel، Microsoft PowerPoint .... الخ او حتى تقنيات اخرى كـ ADO، DAO، DirectX .... الخ، تلاحظ ان لكل منتج او عنصر من هؤلاء اهرام كائنية خاصة به مرتبط بعضها ببعض.
بناء اهرامك الكائنية الخاصة بك امر في غاية الاهمية وكل ما يلزمك هو التركيز في تصميم الهرم الكائني وليس في اكواده، فالتصميم الجيد هو العامل الرئيس لنجاح هرمك الكائني، اما اكواده فتأتي في المرحلة التالية. مع ذلك، اساليب التصميم واعداد المخططات الاولية لانشاء الهرم الكائني خارج نطاق الكتاب، ولكن سأجهزك هنا بكل ما تحتاجه لبناء الاهرام الكائنية وسأبدا بالعلاقات بين الفئات.

العلاقات بين الفئات

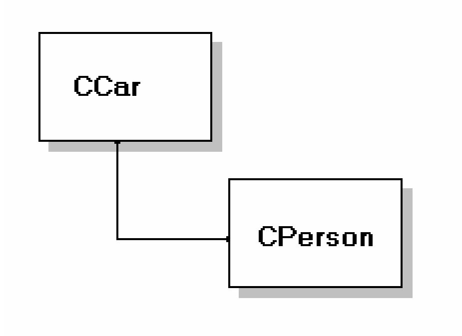
في عالم البرمجة كائنة التوجه OOP يوجد نوع من العلاقات تسمى "يحتوي على Has a" وقد ذكرتها في فقرة "الوراثة" في هذا الفصل، الهدف من هذه العلاقة هو ربط الفئات بعضها ببعض حتى تتمكن من بناء هرم كائني. طريقة الربط تتم بسهولة شديدة، فكل ما عليك القيام به هو تعريف كائن من فئة اخرى في داخل الفئة حتى تصل الى اعضاء الفئة الاخرى. يوجد نوعين من علاقة "يحتوي على" هما:

علاقة 1 الى 1:

علاقة 1 الى 1 هي علاقة بين فئة س وفئة ص تربطها كائنات موجودة في الفئة س لتصل الى اعضاء الفئة ص، فمثلا لو كان لدينا الفئة CCar واردنا ربطها بالفئة CPerson بحيث تمثل مالك السيارة وسائق السيارة فقد تضيف خاصتين objOwner و objDriver في الفئة CCar وتكتب شيئا مثل:

```
' الفئة CCar
Public objOwner As CPerson
Public objDriver As CPerson
Public sCarModel As String
```

وبها تكون قد كونت علاقة في الهرم الكائني البسيط جدا **شكل 6-1**.



شكل 6-1: هرم كائني بسيط يبين العلاقة بين كائناته.

طبعا اذا كان الهرم الكائني يحتوي على اكثر من 200 فئة –مثل مكتبة MFC، فسيكون التصميم كما في الشكل السابق امر في غاية الاهمية ويصبح تركيزك على التصميم اكثر من انجاز الاكواد. اما اذا اردت استخدم الفئات CCar و CPerson فقد تكتب شيئا مثل:

```
Dim Turki As New CPerson
Dim Abbas As New CPerson
Dim BMW As New CCar

Turki.sName = "تركي العسيري"
Abbas.sName = "عباس السريع"

…
…

BMW.sCarModel = "BMW - 7"
Set BMW.objOwner = Turki
Set BMW.objDriver = Abbas
```

بل يمكنك الوصول الى اعضاء الكائنات objOwner و objDriver وانت في داخل الفئة CCar، فقد تضيف طريقة الى الفئة لطباعة اسم المالك والسائق:

```
' تعريف طريقة في الفئة CCar
Public Sub PrintRelatedPeople ( frmForm As Form)
        frmForm.Print "الموديل: " & Me.sCarModel
        frmForm.Print "المالك: " & Me.objOwner
        frmForm.Print "السائق: " & Me. objDriver
End Sub
```

فعلا، البرمجة كائنية التوجه OOP برمجة ممتعة للغاية واقرب الى العالم الحقيقي من البرمجة الاجرائية المعقدة.

علاقة 1 الى ن:

قد تكون هناك اكثر من علاقة بين نفس الكائنات لتمثل علاقة اكثر تعقيدا تعرف بعلاقة 1 الى ن، حيث يحتوي الكائن على مجموعة كائنات من نفس النوع. فلو عدنا الى المثال السابق، سنلاحظ ان السيارة الواحدة CCar لها مالك واحد objOwner و سائق واحد objDriver، نستطيع تطوير كائن السائق بحيث يمكن للسيارة ان يكون لها اكثر من سائق –ولد نعمة! قد تستخدم اسلوب المصفوفات لتعيد تعريف الكائن objDriver كما في الكود التالي:

```
' الفئة CCar
Private m_objDrivers (5) As CPerson

Public Property Get objDriver( iDriverNum As Integer ) As CPerson
        Set objDriver = m_objDrivers ( iDriverNum )
End Property

Public Property Set objDriver( iDriverNum As Integer, ByVal objNewValue As CPerson)
        Set m_objDrivers ( iDriverNum ) = objNewValue
End Property
```

بامكانك الوصول الى هذه الخاصية بكتابة شيئا مثل:

```
Set BMW.objOwner = Turki
Set BMW.objDriver (0) = Abbas
Set BMW.objDriver (1) = Ahmed
Set BMW.objDriver (2) = Ali
...
```

...

يعرف النوع السابق بانه "علاقة 1 الى ن مبنية على المصفوفات" Array based 1 to many relationship، ويوجد نوع اخر –افضله كثيرا- يعرف "بعلاقة 1 الى ن مبنية على المجموعات" Array based 1 to many relationship، حيث تكون الخاصية objDriver عبارة عن مجموعة Collection بدلا من مصفوفة:

```
' الفئة CCar
Public objDrivers As New Collection
```

بامكانك اضافة الكائنات الى الخاصية objDriver مباشرة:

```
BMW.objDrivers.Add Turki
BMW.objDrivers.Add Ali
```

او حتى الوصول الى كافة عناصر الخاصية objDriver باستخدام حلقة For … Each:

```
Dim objDriver As CPerson

For Each objDriver In BMW.objDrivers
   Print objDriver.sName
Next
```

المشكلة في العلاقات المبنية على المجموعات تظهر عندما نعلم ان المجموعة (الخاصية) objDriver يمكنها ان تحمل أي نوع من القيم، فهذا الكود سيتم تنفيذه بشكل صحيح:

```
Dim BMW As New CCar
Dim Mercury As New CCar

…
…

BMW.objDrivers.Add Mercuty          ' سائق السيارة هي سيارة اخرى!
```

والحل يتم بانشاء فئة خاصة تسمى فئة المجموعة CPersons Collection Class لها طرق وخصائص قياسية يتبعها كل المبرمجين وجميع المكتبات والكائنات المتوفرة في Visual Basic و لغات البرمجة الاخرى المتوافقة مع COM كما في الفقرة التالية.

فئات المجموعات Collection Classes

فئات المجموعات ما هي الا فئات عادية لكن لها خصائص وطرق قياسية عليك اتباعها حتى يطلق على الفئة اللقب "فئة مجموعة" Collection Class، وان اصررت على عدم اتباع هذه المواصفات القياسية للفئة، فارجو ان تعود الى رشدك وتلين عنادك قليلا، فجميع الفئات المنجزة بلغات البرمجة المختلفة والداعمة لتقنية COM تتبع هذا الاسلوب بما فيهم Visual Basic.
اول قاعدة عليك معرفتها هي ان فئة المجموعة تمثل مجموعة لكائنات معينة ونمييزها عن فئة الكائنات بالحرف "s"، فالفئة CPerson يمكن ان تكون تابعة لفئة مجموعة باسم CPersons.
اما لبناء المجموعة CPersons انشئ فئة جديدة وسمها CPerons واكتب هذا الكود المحاكي لطرق وخصائص المجموعات Collections:

```
' فئة المجموعة CPersons
Private m_Col As New Collection

Public Sub Add(objNewItem As CPerson, Optional vKey As Variant, _
Optional vBefore As Variant, Optional vAfter As Variant)
        m_Col.Add objNewItem, vKey
End Sub

Public Sub Remove(vIndexKey As Variant)
        m_Col.Remove vIndexKey
End Sub

Public Property Get Count() As Long
        Count = m_Col.Count
End Property

Public Property Get Item(vIndexKey As Variant) As CPerson
        Set Item = m_Col(vIndexKey)
```

```
End Property
```

ملاحظة: تستطيع استخدام الاضافة Add-Ins المسماة Class Builder Utility لتسهيل عملية كتابة طرق وخصائص فئة المجموعة تلقائيا ومن ثم تعديلها بنفسك.

بامكانك الانتقال الى الفئة CCar وتغيير نوع الكائن objDrivers من Collection الى CPersons:

```
Public objDrivers As New CPersons
```

والان استخدم الفئات CCar و CPerson بنفس الطريقة السابقة وكأن شيئا لم يحدث:

```
Dim BMW As New CCar
Dim Driver1 As New CPerson
Dim Driver2 As New CPerson
…
…

Driver1.sName = "محمد"
Driver2.sName = "عبدالله"
…
…

BMW.objDrivers.Add Driver1
BMW.objDrivers.Add Driver2
```

ضرورة الالتزام بالمعايير القياسية:

في بداية الفقرة اكدت على مسألة الالتزام بالمواصفات والمعايير القياسية لفئات المجموعات، وقد قمنا بعملية محاكاة لمعظم طرق وخصائص المجموعات كما ينبغي، ولكن بقيت نقطتين بودي توضيحها، الاولى تتعلق بالخاصية Item التابعة للمجموعة CPersons، فيجب ان تكون الخاصية الافتراضية Default Property للمجموعة CPersons بحث يمكن للمبرمج تجاهلها:

```
Print BMW.objDrivers.Item(1).sName
Print BMW.objDrivers(2).sName
```

بامكانك عمل ذلك عن طريق صندوق الحوار Procedure Attributes كما ذكرت في الفصل السابق "البرمجة كائنية التوجه OOP".

اما النقطة الثانية التي اود ان اذكرها هي قابلية استخدام الحلقة For ... Each مع المجموعة CPersons، فلو كتبت هذا الكود:

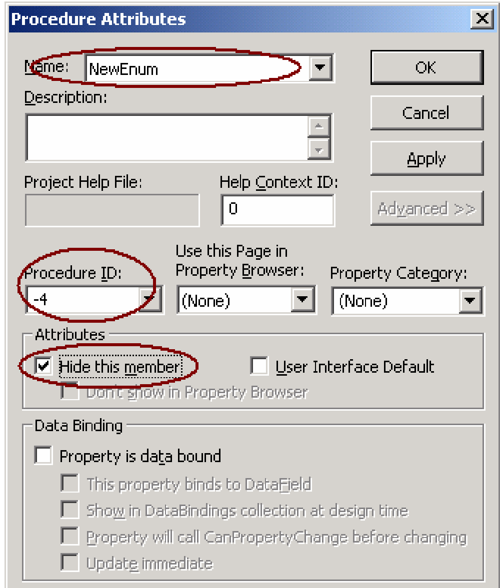
```
Dim objDriver As CPerson

For Each objDriver In BMW.objDrivers
   Print objDriver.sName
Next
```

سيظهر لك Visual Basic رسالة خطأ، لان الحلقة For ... Each ليست مدعومة في مجموعتنا الجديد CPersons، لذلك عليك كتابة الكود التالي في المجموعة CPersons:

```
Public Property Get NewEnum() As IUnknown
        Set NewEnum = m_Col.[_NewEnum]
End Property
```

وكتابة القيمة 4- في الخانة Procedure ID التابعة للاجراء NewEnum في صندوق الحوار Procedure Attributes وتحديد الاختيار Hide this member **شكل 6-2**، وتكون بذلك قادر على استخدام الحلقة For ... Each مع المجموعة CPersons.

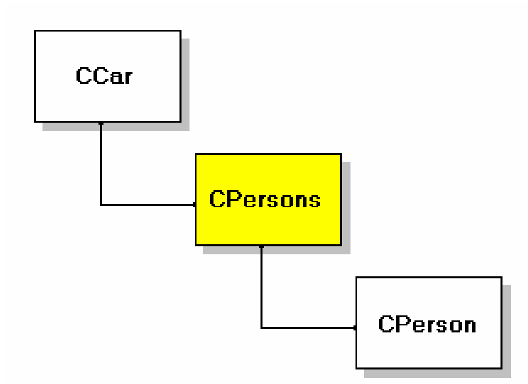


شكل 6-2: تعديل مواصفات الاجراء NewEnum.

ملاحظة: استيعاب الخطوات السابقة خارج نطاق الكتاب، لانه يتطلب فهم البنية التركيبات لمكونات COM التابعة لـ OLE Automation والخاصة بواجهات المكونات كالواجهة IUnknown او بالتحديد الواجهة IEnumVariant. اذا اردت مزيد من التفاصيل المتقدمة حول هذا الموضوع انصحك بكتاب:

Advanced Visual Basic 6
Power Techniques for Everyday Program
By: Matthew Curland
ISBN: 0-201-70712-8

اخيرا، اذا اردت اعادة رسم الهرم الكائني CPerson، CPersons و CCar فيفضل استخدام لون يميز فئات المجموعات عن الفئات العادي **شكل 6-3** فهو الاسلوب المتبع في ملفات التعليمات ومواقع الانترنت –التي رأيتها.



شكل 6-3: الهرم الكائني بعد اضافة المجموعة CPersons.

بهذا اكون قد انتهيت من تشييد بنية اساسية لتكون مبـرمج Visual Basic حقيقـي بعدما تطرقت الى المبادئ والاساسيات التـي لابـد علـى كـل مبـرمج Visual Basic مـن معرفتهـا واتقانهـا للابحـار فـي برمجـة Visual Basic. وهـذه نهايـة الجـزء الاول "الاساسيات" من هذا الكتاب، والان بامكانك تعليم نفسك ذاتيا امـا بالحصـول علـى كتب متخصصة في مجال معين، او قراءة مقـالات متقدمـة، او حتـى الاسـتمرار فـي قراءة هذا الكتاب ان كان اسلوب المؤلف ليس سيئا ومناسبا لك.